НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»
Факультет Інформатики та Обчислювальної техніки
(повна назва інституту/факультету)

Кафедра Обчислювальної техніки
(повна назва кафедри)

До захисту допущено
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
«___» _____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія программного забезпечення»

на тему: «СИСТЕМА КЕРУВАННЯ КОНТЕНТОМ**»**

Виконав (-ла):
студент (-ка) IV курсу, групи ІП-62
Литвинчук Данило Кирилович _____

Керівник:
к.т.н., доцент
Корочкін Олександр Володимирович _____

Консультант з нормконтролю:
д.т.н., професор
Сімоненко Валерій Павлович _____

Рецензент:
доц. каф. СПСКС
Оксана Тарасенко-Клятченко _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень **бакалавр**

Напрям підготовки **121 «Інженерія программного забезпечення »**

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“___”_____2020 р.

ЗАВДАННЯ

на бакалаврський дипломний проект студента

Литвинчука Данила Кириловича
(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) *Система керування контентом*,

керівник проекту (роботи) Корочкін Олександр Володимирович к.т.н., доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом по університету від “ 07 ” травня 2020 р. № 1081-с

2. Термін здачі студентом закінченого проекту (роботи) 20 червня 2020р.

3. Вихідні дані до роботи технічна документація програмного комплексу системи, і програмна реалізація системи, технології використані для реалізації системи керування контентом – бібліотека React.js.

4. Зміст пояснювальної записки: *опис предметної області, розгялд існуючих рішень, реалізація программного комплексу.*

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): презинтація до дипломного проекту, діаграма класів, структурна схема компонентів.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	д.т.н., проф. Сімоненко В. П.		

7. Дата видачі завдання 10 лютого 2020р.

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1.	Затвердження теми роботи	10.12.2019-18.01.2020	
2.	Вивчення та аналіз завдання	15.01.2020-15.03.2020	
3.	Написання вступної частини та огляду предметної області	15.03.2020-31.03.2020	
4.	Розробка архітектури системи	01.04.2020-10.04.2020	
5.	Програмна реалізація системи	11.04.2020-30.04.2020	
6.	Виправлення помилок	01.05.2020-10.05.2020	
7.	Оформлення документації дипломної роботи	11.05.2020-26.05.2020	
8.	Передзахист	27.05.2020	
9.	Захист	18.06.2020	

Студент _____ Данило ЛИТВИНЧУК
(підпис) (ініціали, прізвище)

Керівник проекту _____ Олександр КОРОЧКІН
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

до бакалаврської дипломної роботи Литвинчука Данила Кириловича на тему:«Система керування контентом»

Дипломна робота присвяча створенню системи керування контентом, яка в свою чергу має ряд переваг перед своїми аналогами перш за все в інтерфейсі користувача та роботою з данними.

Система може виконувати наступні задачі:

1) Відображати данні з певного сховища данних
2) Сортувати та фільтрувати данні
3) Редагувати данні

В роботі використано новітню технологію для побудови оптимізованих інтерфейсів користувача – бібліотеку React.js. Запропоновано використання програмного забезпечення Docker, що в свою чергу надає можливість кросплатформеного запуску.

АННОТАЦИЯ

к бакалаврской дипломной работы Литвинчука Даниила Кирилловича на тему: «Система управления контентом»

Дипломная работа посвященна созданию системы управления контентом, которая в свою очередь имеет ряд преимуществ перед своими аналогами в первую очередь в интерфейсе пользователя и работой с данными.

Система может выполнять следующие задачи:

1) Отображать данные с определенного хранилища данных

2) Сортировать и фильтровать данные

3) Редактировать данные

В работе использованы новейшую технологию для построения оптимизированных интерфейсов - библиотеку React.js. Предложено использование программного обеспечения Docker, что в свою очередь дает возможность кроссплатформенного запуска.

ANNOTATION

to the bachelor's thesis of Lytvynchuk Danylo
Kyrylovych on the topic: “Content Management System”

The thesis is dedicated to the creation of the content management system, which in turn has a number of advantages over its counterparts, primarily in the user interface and working with data.

The system can perform the following tasks:

1) Display data from a specific data warehouse

2) Sort and filter data

3) Edit data

The work uses the latest technology for building optimized interfaces - the React.js library. The use of Docker software is proposed, which in turn makes it possible to cross-platform launch.

Технічне завдання
до дипломного проекту

на тему: «Система керування контентом»

Київ – 2020

ЗМІСТ

					ІАЛЦ.467800.003 ТЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Литвинчук Д.К.			Система керування контентом Технічне завдання	Літ.	Аркуш 1 / Аркушів 4
Перевірив		Корочкін О.В.					
Реценз.						НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІП-62	
Н. Контр.		Сімоненко В.П.					
Затвердив							

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання стосується розробки програмно забезпечення по системі керування контентом. Областю застосування всієї системи є Data Analysis, Data Science, допомога у вивченні інших дисциплін пов'язаних із обробкою данних.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврської дипломної роботи, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут» імені Ігоря Сікорського.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є система керування контентом найрізноманітнішої направленності.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з теорії і практики програмування, бакалаврські роботи інших студентів, публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до апаратно-програмного продукту, що розробляється

Система, що розроблюється, повинна вміти:

1) Відображати данні з певного сховища данних

2) Сортувати та фільтрувати данні

3) Редагувати данні

5.2. Вимоги до апаратної частини обчислювальної системи оператора

- одноядерний процесор з тактовою частотою не менше ніж 1 ГГц;

- оперативна пам’ять об’ємом не менше ніж 512 Мб;

- жорсткий диск об’ємом не менше ніж 20 Гб;

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2019-15.12.2019
Вивчення та аналіз завдання	15.12.2019-15.03.2020
Написання вступної частини та огляду предметної області	15.03.2020-31.03.2020
Розробка архітектури системи	01.04.2020-28.04.2020
Програмна реалізація системи	29.04.2020-10.05.2020
Виправлення помилок	11.05.2020-15.05.2020
Оформлення документації дипломної роботи	16.05.2020-26.05.2020
Передзахист	27.05.2020
Захист	18.06.2020

Відомість
дипломного проекту

на тему: «Система керування контентом»

Київ – 2020

№ рядка	Формат	Позначення	Найменування	Кільк. аркуші	Примітка
1			Документація		
2					
3	А4		Завдання	2	
4			на дипломний проект		
5					
6	А4	ІАЛЦ.466521.002 ВП	Відомість	1	
7			дипломного проекту		
8					
9	А4	ІАЛЦ.466521.003 ТЗ	Технічне завдання	4	
10					
11	А4	ІАЛЦ.466521.004 ПЗ	Пояснювальна записка	51	
12					
13	А4	ІАЛЦ.466521.005 Д1	Додаток 1. Лістинг програми	12	
14					
15	А4	ІАЛЦ.466521.006 Є1	Блок-схема алгоритму	1	
16					
17	А4	ІАЛЦ.466521.007 Д2	Діаграма класів компонентів	1	
18					
19	А4	ІАЛЦ.466521.008 Д3	Діаграма класів запитів	1	
20					
21					
22					
23					
24					
25					
26					

Зм	Аркуш	№ докум	Підпис	Дата	ІАЛЦ.466521.002 ВП		
Розробив		Литвинчук Д.К.			Система керування контентом	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Корочкін О.В.					
Т. контр.					Відомість дипломного проекту	НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІП-62	
Н. контр.		Сімоненко В.П.					
Затверд.		Стіренко С.Г					

Пояснювальна записка
до дипломного проекту

на тему: «Система керування контентом»

Київ – 2020

ЗМІСТ

ВСТУП

Мета данного дипломного проекту — розробка системи керування контентом (CMS). CMS система вирішує загальні проблеми забезпечення і організації спільного процесу створення, редагування і керування контентом.

Якщо раніше більшість сайтів були статичними і вимагали внесення правок в їх контент вручну, то зараз динаміка розвитку проектів вимагає готовності швидко реагувати на зміни і впроваджувати їх з максимальною оперативністю. При цьому не всі користувачі хочуть або можуть собі дозволити звертатися до розробників, особливо якщо сайт вимагає постійної роботи над ним.

Запропонована в роботі система ставить перед собою мету вирішити проблеми простоти та доступності данних.

Для кожного користуваяа існують свої унікальні вимоги до CMS, і єдиного списку вимог виробити майже неможливо, проте дуже важливо щоб система була спроможна надавати:

- інструменти для створення контенту
- організацію спільної роботи над контентом
- обробка вмісту: управління потоком данних
- публікація вмісту
- подання інформації у вигляді, зручному для навігації, пошуку

В проекті виконано розробку відповідного програмного забезпечення на основі новітньої технології – React.js, у якості сховища данних для запропонованої системи керування контентом може бути використано будь-яке сховище данних: SQL або NOSQL база данних, або навіть текстовий файл.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Опис завдання

Завдання дипломної роботи полягає у розробці системи керуванням контентом, яка в свою чергу надасть своїм користувачам легкі та зручні інструменти для створення контенту, організує спільну роботу над контентом, надасть можливість обробки контенту, його зберігання, обробки доступу та публікацію вмісту.

Аналіз отриманих результатів дозволить користувачам, які не володіють навичками розробки сайтів і знаннями мов програмування, самостійно працювати над створенням і зміною контенту сайту.

1.2 Аналіз існуючих рішень

Наразі існує безліч готових систем керування контентом сайту, в тому числі і безкоштовних. Тим не менше, більшість CMS гнучко настроюється і можуть бути використані для розробки сайтів різної спрямованості. Наприклад, найбільш популярним і універсальним варіантом є WordPress, на якому можливо створити практично будь-який проект: від особистого сайту до великого порталу або інтернет-магазину. Існують також CMS, розроблені на замовлення під конкретний проект. Як наслідок, їх функціонал не так широкий, ніж у масових систем, але максимально відповідає поставленим завданням і не містить зайвих інструментів. Далі розглянемо детальніше наступні готові рішення:

- Система WordPress [8]
- Продукт компанії Shopify [11]
- Система Drupal [9]
- Система Joomla [8]

1.2.1 Система WordPress

WordPress – система керування контентом сайту, написана на PHP, та використовує сервер бази даних - MySQL, випущена під ліцензією GNU GPL

версії 2 [6]. Може бути використана від блогів до досить складних новинних ресурсів. Вбудована система плагінів разом з вдалою архітектурою дозволяє конструювати проекти широкої функціональної складності.

Зазвичай, ця CMS використовується для створення блогу, але сайт на WordPress може бути легко перетворений в інтернет магазин, портфоліо, сайт періодичного характеру або що-небудь інше.

Перш ніж приступати до створення сайту на WordPress, необхідно вибрати відповідний хостинг і придбати домен. Деякі хостери пропонують спеціальні тарифи, оптимізовані для роботи з движком, але купувати їх не обов'язково - для першого сайту цілком підійде звичайний віртуальний хостинг (рис.1.1):

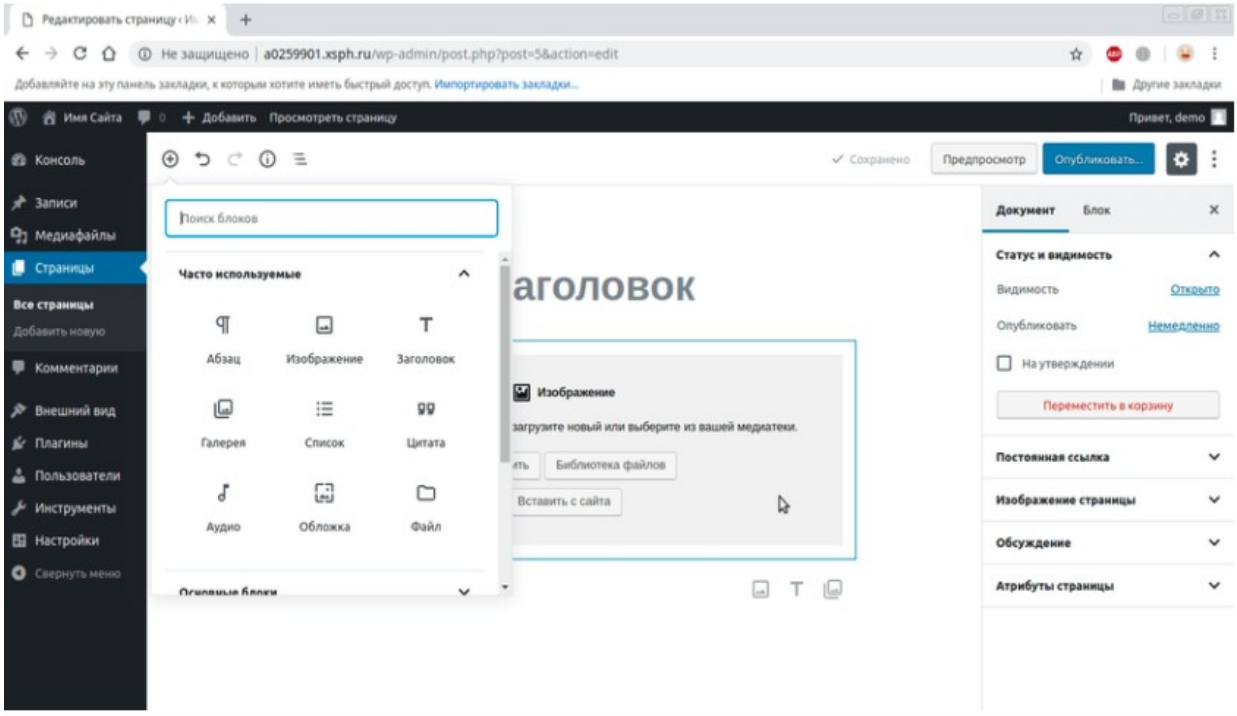


Рис. 1.1 Звичайний віртуальний хостинг

Усередині кожного розділу представлені тематичні підрозділи (рис 1.2), що містять власні набори налаштувань. Все це робить систему максимально гнучкою:

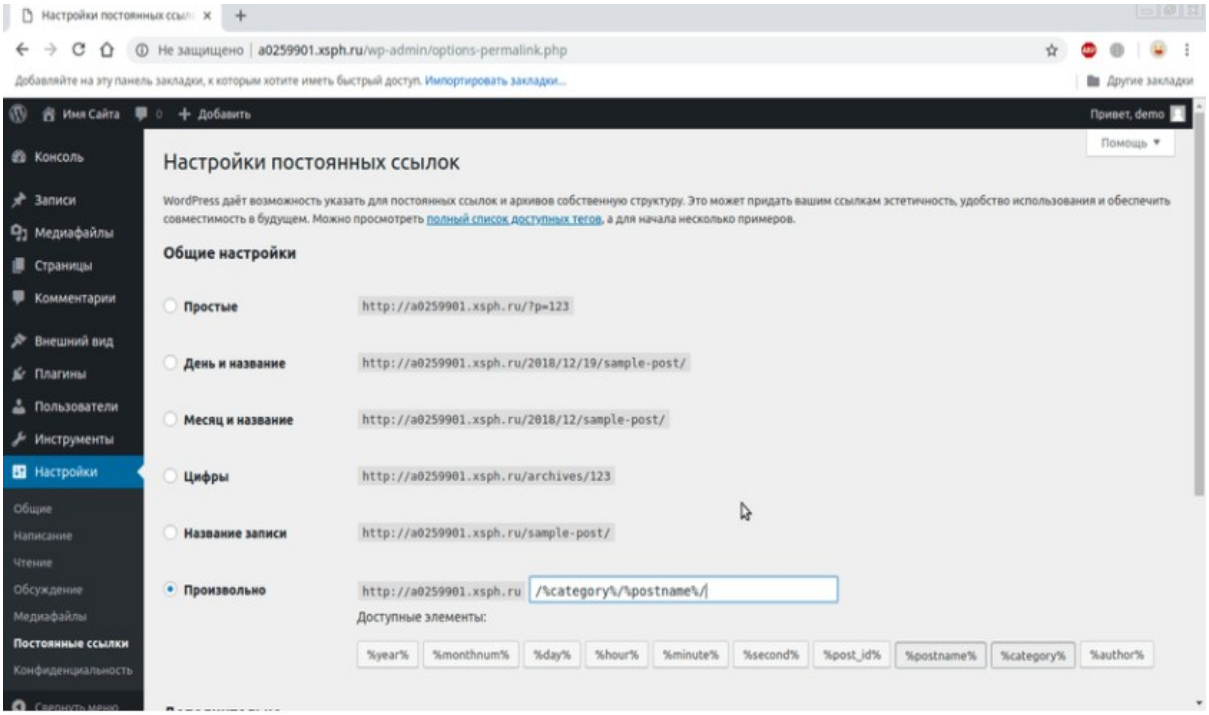


Рис. 1.2 Тематічні підрозділи

Кожен блок можна налаштовувати окремо (рис 1.3). Для додавання віджету на сторінку або поста в запис не потрібно працювати з кодом: досить вибрати відповідний блок і вказати посилання на контент, після чого він з'явиться в полі редактора:

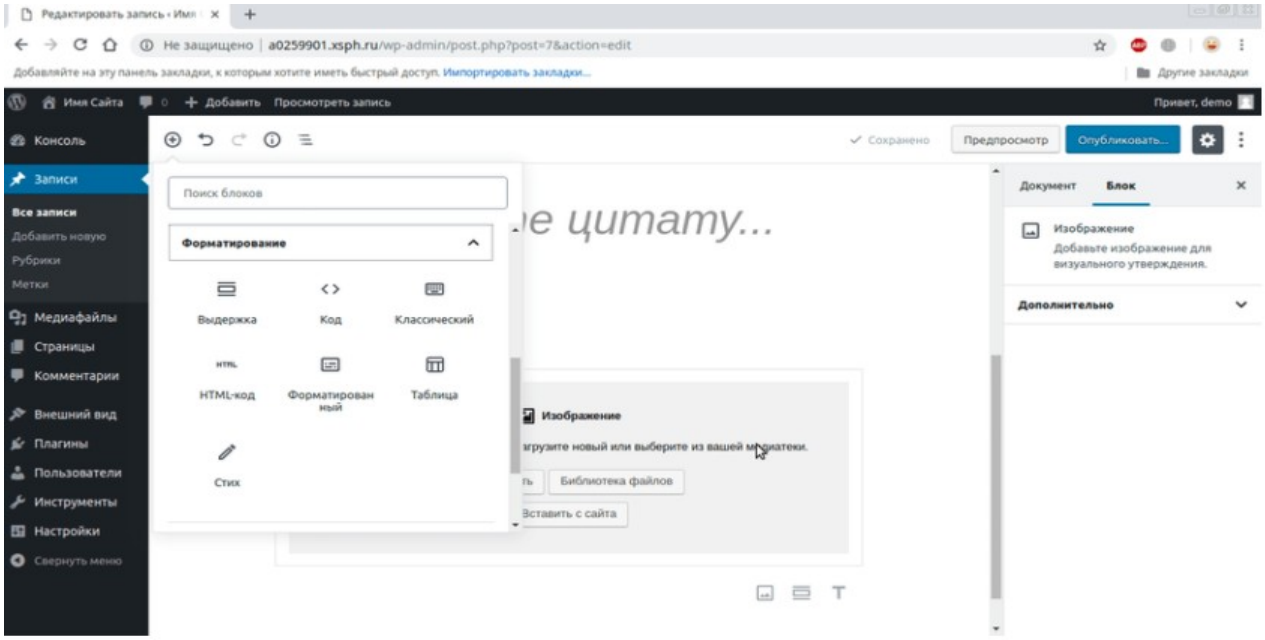


Рис. 1.3 Налаштування окремого блоку

Сайти будуються на основі шаблонів, яких для WordPress розроблена величезна кількість. Стандартний каталог (рис 1.4) тим доступний з панелі управління движка в розділі «Зовнішній вигляд». Тут можна вибрати і встановити шаблон з бібліотеки, а також завантажити файли, викачані з інших джерел:

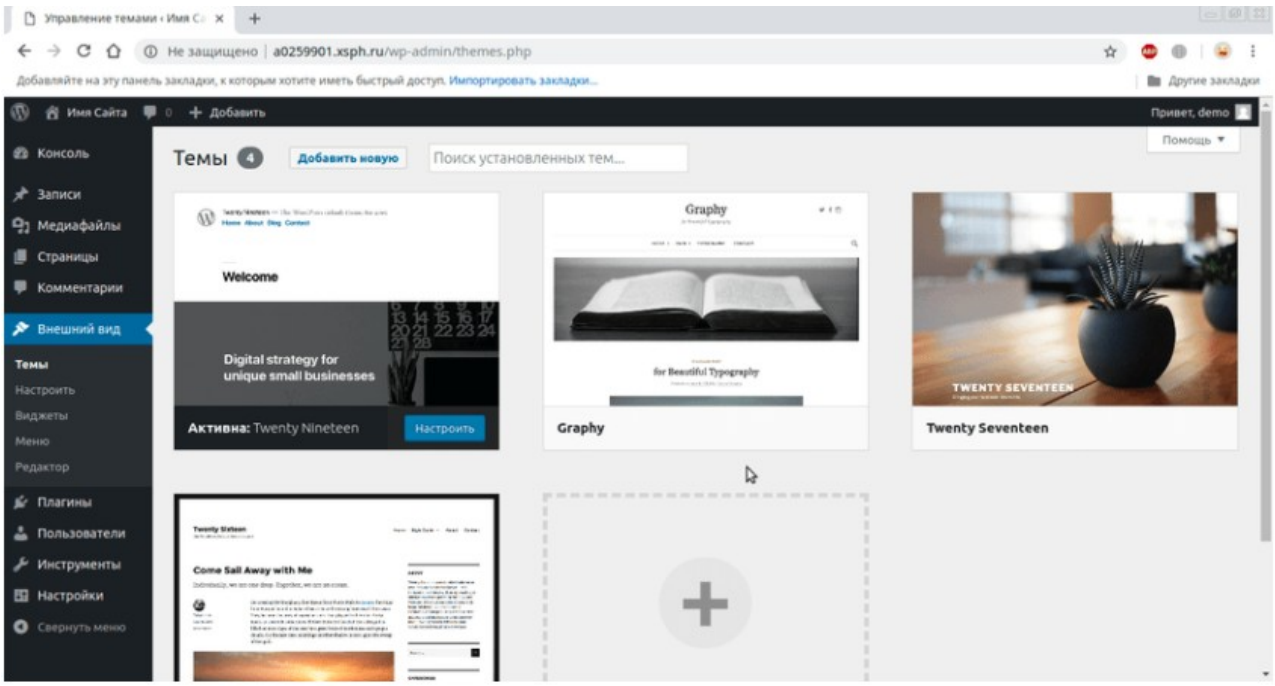


Рис. 1.4 Стандартний каталог шаблонів

Отже, серед основних переваг WordPress можна відзначити:

- Проста панель управлінн. Не потрібні знання з програмування для виконання повсякденних завдань, таких як написання та редагування публікацій, завантаження та редагування зображень, керування користувачами, додавання меню та встановлення плагінів та тем.
- Редактор, що надає нові можливості для зміни інтерфейсу сайту і управління записами в візуальному режимі
- Величезна кількість шаблонів і розширень, що роблять WordPress універсальною

- Потужна база знань і розвинене співтовариство користувачів
- Постійні апдейти і поява додаткових матеріалів движка

  Низька вартість. Потрібно заплатити лише за домен та послугу веб-хостингу. Програмне забезпечення WordPress та багато плагінів та теми безкоштовні.
- Індивідуальний дизайн - Завдяки тисячам готових до використання тем WordPress можна легко створити індивідуальний дизайн. Наприклад, є спеціальні теми для ресторанів, медиків, малого бізнесу, продовольчих блогерів тощо.
- Спеціальна функціональність плагінів. Можна використовувати плагіни для розширення функцій WordPress за замовчуванням. Можна знайти плагін для кожного конкретного завдання - від оптимізації пошукової системи до бронювання подій.
- Низька вартість

Серед недоліків:

- Зниження продуктивності через використання великої кількості плагінів
- Безпека. Оскільки WordPress має приблизно більше 35% Інтернету, на нього також часто націлюються хакери. Однак якщо встановити плагін безпеки, можна зменшити ризик.
- Сторонній контент. Оскільки більшість плагінів і тем створені сторонніми розробниками, вони іноді бувають помилковими. Перш ніж встановлювати новий плагін або тему, потрібно завжди читати опис та огляди.
- Час завантаження сторінки - якщо у додатку занадто багато плагінів, сайт може стати повільним. Однак зазвичай проблема вирішення цієї проблеми може бути встановлена плагіном кешування.

1.2.2 Продукт компанії Shopify

Shopify - це провідна платформа, що дозволяє підприємцям створювати власні інтернет-магазини. Shopify проста у використанні, тому з її допомогою можна створити будь-яку платформу для покупок. Вона ідеально підходить для підприємців, які хочуть запускати власний інтернет-магазин без будь-яких турбот і великих фінансових витрат.

Панель управління в Shopify (рис 1.5) насичена різноманітними налаштуваннями, не надто проста, але цілком дружелюбна по відношенню до користувача:

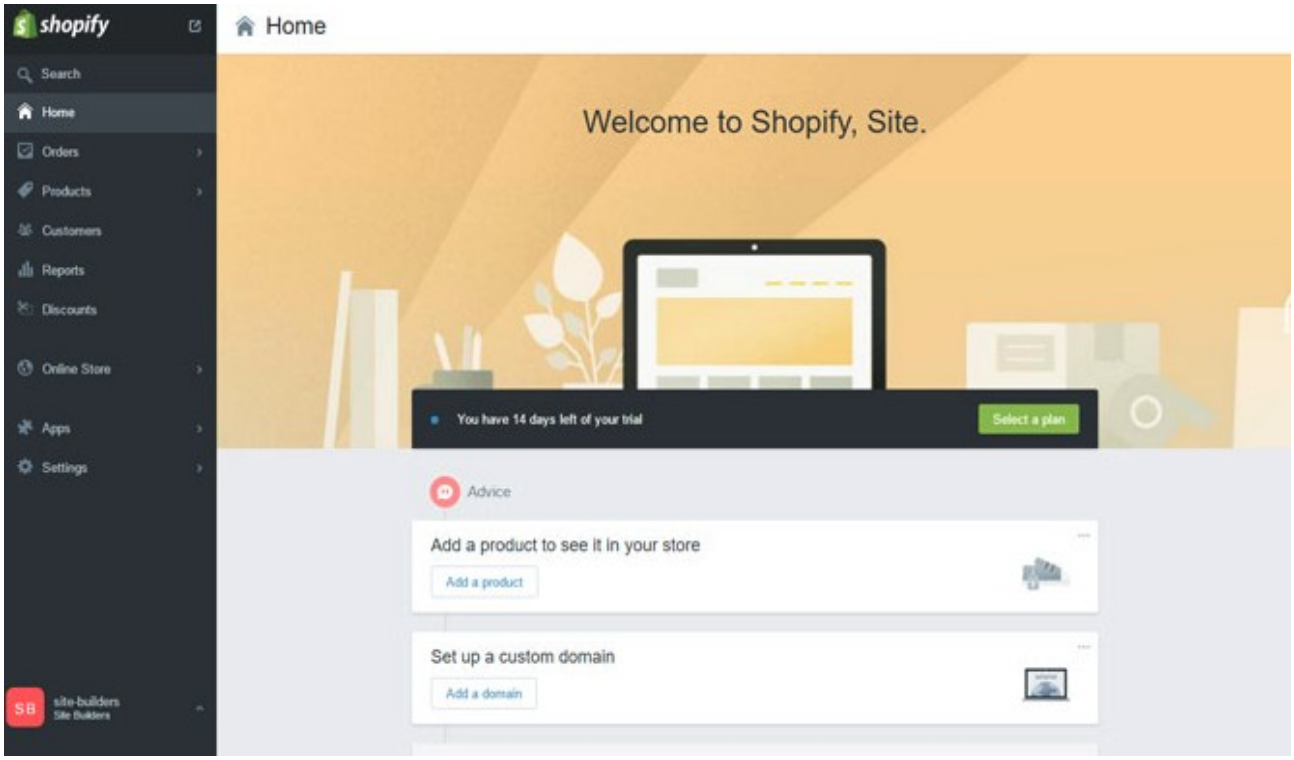


Рис 1.5 Панель управління

Окремо варто сказати про можливість завантаження / розвантаження данних через CSV-файли (рис 1.6). Можна завантажити зразок, який допоможе правильно оформити, а потім завантажити файл:

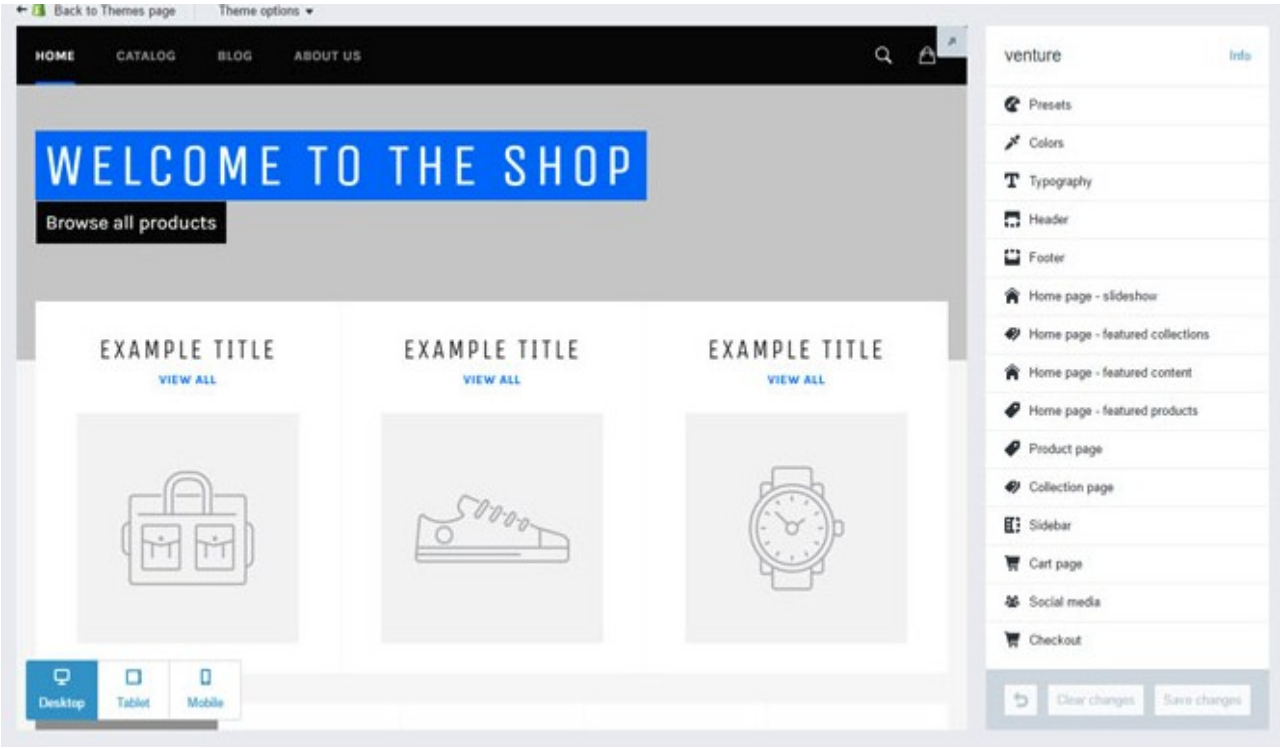


Рис 1.6 Завантаження товарів через CSV-файли

Всі шаблони адаптивні. Можливості по їх кастомізації в Shopify дуже великі, причому мають індивідуальні відмінності в залежності від обраного зразка. Можливість редагування коду шаблонів (рис 1.7) розв'язує руки досвідченим користувачам:

Рис 1.7 Редагування шаблонів

Блоги, сторінки та навігація по сайтах, все це керується через інтуїтивно зрозумілі екрани адміністрування. Повнофункціональна система активів, яка дозволяє використовувати та використовувати фотографії, логотипи або PDF-файл продукту, де вони вам потрібні. Можливо навіть завантажити вихідний код обраної теми , або попрацювати в місцевому редакторі. Проте у всіх шаблонах використовується англійська мова. Локалізувати доведеться вручну (рис 1.8):

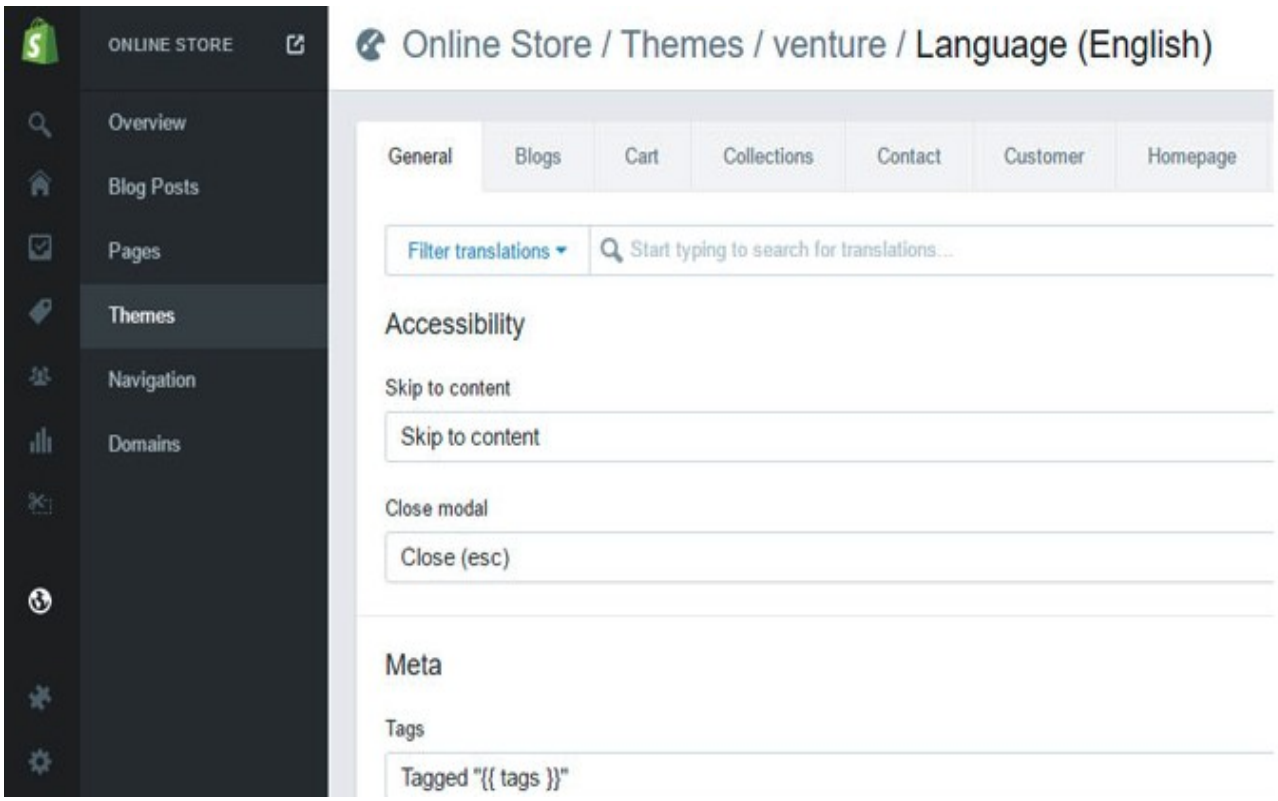


Рис 1.8 Локалізація контенту

Shopify відрізняється від інших аналогічних електронних комерційних рішень в основному завдяки наступному функціоналу:

- В Shopify налаштувати та розпочинати роботу надзвичайно просто, тому його можуть використовувати початківці, які не мають технічних знань щодо створення та роботи сайту.
- Весь інструмент дуже доступний (детальніше про це за хвилину), тому це привабливий вибір для малого бізнесу, який тільки починається і хоче

мінімізувати витрати.

- Безліч шаблонів конструкцій / структур, якими можна скористатися, тому вам доведеться наймати дизайнера чи когось, хто допоможе у магазині. Можна буквально створювати, створювати стиль та запускати свій інтернет-магазин та систему керування контентом, все самостійно.

Отже, серед плюсів Shopify відзначимо:

- Загальний високий рівень функціональності, панелі керування магазином
- Безліч шаблонів, гнучкі засоби. Оформлення та структури кожної сторінки
- Легко додавати, перелічувати, редагувати та організовувати
- Можливо легко розширити функціональність Shopify завдяки величезному набору сторонніх додатків.
- Справді добре справляється зі створенням та застосуванням контенту
- Поставляється із вбудованим блогом

В свою чергу недоліки:

- Бібліотека додатків не бідна, але половина асортименту марна
- Необхідно вручну локалізувати шаблон
- З коробки немає можливості синхронізації з платіжними засобами
- Деякі основні функціональні можливості, які вимагають встановити додаток
- Додавання користувальницьких полів, таких як текстові поля або параметри завантаження файлів є надмірно складним (або передбачає придбання програми)
- Немає можливості автоматично переконатися, що зображення товарів відображаються за допомогою однакового співвідношення сторін.
- Професійний функціонал звітності надається лише на більш дорогих рівнях
- Експорт публікацій блогів із Shopify не є таким простим, як це має бути.
- Не можна уникнути плати за транзакції для стороннього шлюзу платежів

1.2.3 Система Drupal

Drupal - гарна CMS з модульною структурою, що дозволяє додавати і видаляти функціонал за дапомогою встановлення і видалення модулів. Також дана CMS дозволяє змінити тему сайту через установку і видалення за допомогою тем оформлення. Основа Drupal, містить PHP скрипти необхідні для запуску основного функціоналу CMS, декількох додаткових модулів (рис 1.10) і тим, і безлічі JavaScript, CSS і файлів зображень:

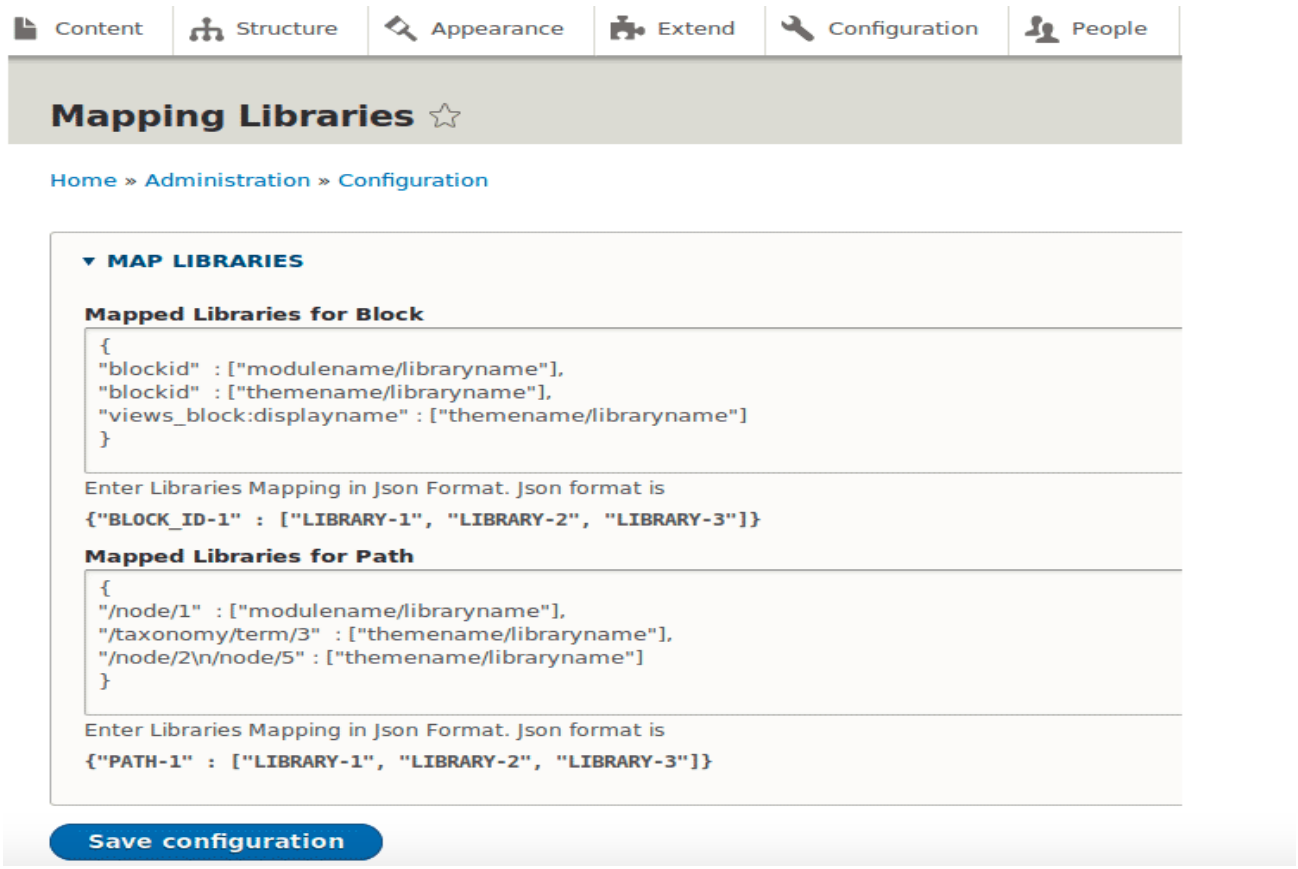


Рис 1.10 Приклад додаткового модулю

Розглянемо найбільш затребувані і зручні функції (рис 1.11): фільтр - зручний інструмент, що дозволяє сортувати матеріали за заданими критеріями; особистий кабінет - після реєстрації, у відвідувача повинен з'явитися аккаунт і вся інформація про нього; каталог - на всіх великих ресурсах існує каталог. Він може складатися з статей, новин, товарів, оглядів, фотографій, відео роликів та інших матеріалів; блоки - це відмінний спосіб розмістити рекламу, важливі новини, схожі матеріали або форму зворотного зв'язку. Адміністратор має блоки

в будь-якому регіоні на свій розсуд:

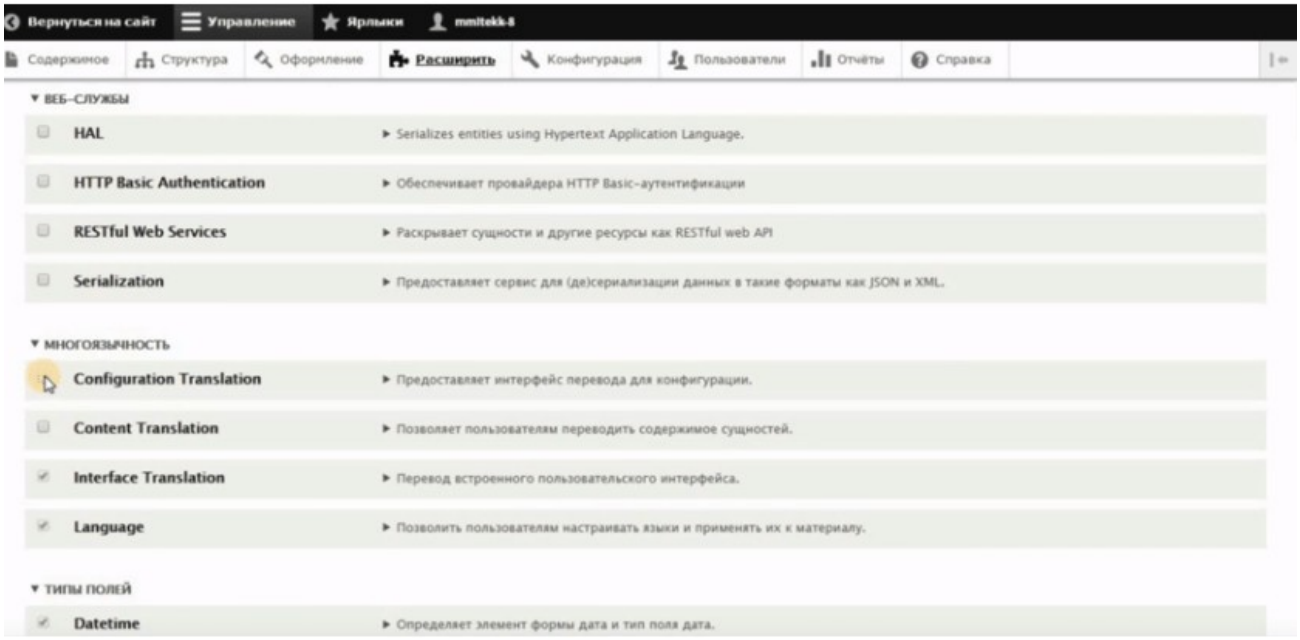


Рис 1.11 Найбільш затребувані і зручні функції

Серед переваг Drupal відзначимо:

- Відкритий вихідний код і структура, що забезпечує максимальну гнучкість системи при налаштуванні
- Проста панель управління, відкрита для кастомізації
- Вбудована система кешування, що забезпечує прискорення завантаження сторінок
- Багатомовність

Недоліки:

- Зазвичай потрібно встановлювати декілька додатків до модулю
- нестабільна робота після оновлень движка
- Надає велику кількість розширень та шаблонів, але доведеться залишити систему для пошуку модулів, перш ніж встановлювати їх. Багато модулів не є безкоштовними, тому доведеться їх придбати.
- Більшість тем веб-сайтів Drupal кодовані на замовлення, що означає, що може знадобитися співпрацювати з розробником, щоб створити відповідний веб-сайт

1.2.4 Система Joomla

Joomla - CMS, написана на мовах PHP і JavaScript, що використовує в якості сховища бази даних СУБД MySQL або інші стандартні промислові реляційні СУБД [2].

Розглянемо найбільш популярні можливості даної платформи (рис 1.12):

- пошук і фільтри. Щоб користувачі могли швидко знайти потрібний контент, товар або послугу, можна встановити рядок пошуку
- текстові та графічні редактори. CMS для сайту дозволяє створювати будь-яку структуру текстів, вирівнювати їх по ширині, додавати марковані списки, таблиці та інші об'єкт

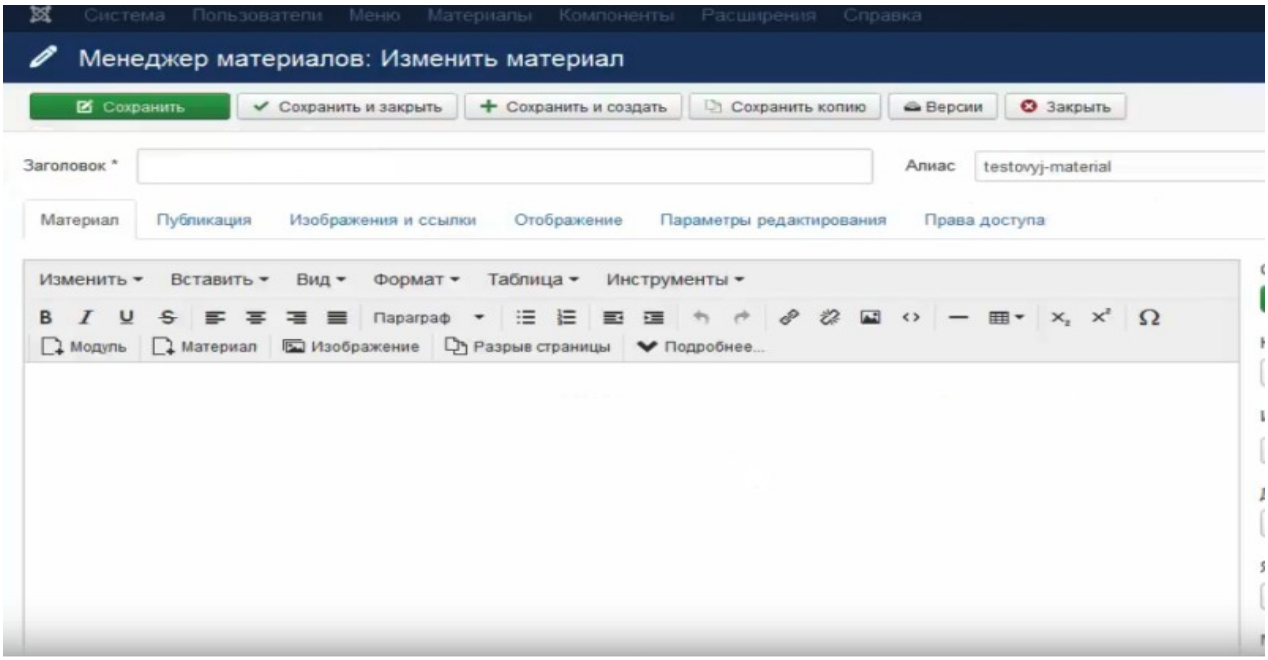


Рис 1.12 Найбільш популярні можливосі Joomla

В даний час платформа налічує близько восьми тисяч доповнень для вирішення різних завдань. Адміністратор може абсолютно безкоштовно встановити (рис 1.13) будь-який з них на свій движок сайту:

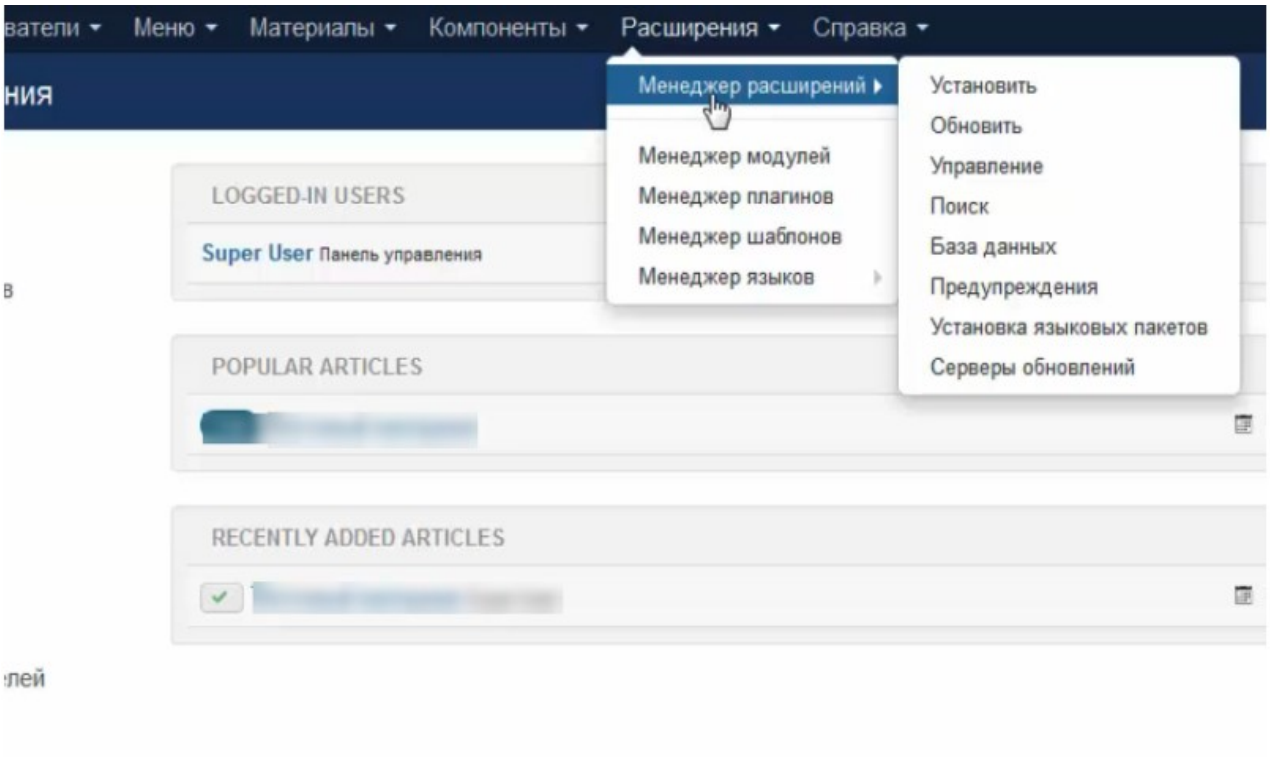


Рис 1.13 Встановлення додатків у Joomla

Таким чином, адміністратор сам вирішує, що повинна вміти його платформа. Можна зібрати систему управління контентом, заточену під створення торгових майданчиків, блогів, новинних порталів і так далі. Тобто, серед переваг Joomla відзначимо:

- Проста установка. На більшості хостингів Joomla розгортається в автоматичному режимі
- Зручна панель адміністратора, на освоєння якої не потрібно багато часу
- Доступність розширень. Плагін або компонент Joomla може надавати різні функції, які не потрібно встановлювати для багатьох плагінів / компонентів при створенні веб-сайту

Серед недоліків:

- Високий відсоток зламаних сайтів
- Несумісність версій движка і плагінів
- Складність. Joomla досить складний, що ускладнює включення власного дизайну без досвіду розробника.

ВИСНОВКИ ДО РОЗДІЛУ 1

В даному розділі було розглянуто існуючі CMS системи. Як ми побачили в системі керування контентом можуть перебувати найрізноманітніші дані: документи, фільми, фотографії, номера телефонів, наукові дані і так далі. Така система часто використовується для зберігання, управління, перегляду і публікації документації. Контроль версій є однією з важливих можливостей, коли вміст змінюється групою осіб.

Системи були порівнянні по різних критеріях, в числі яких: графічний інтерфейс, гнучкість відносно структури вхідних даних, наявність потрібних інструментів в одному сервісі.

Найкращі результати у наявності функціоналу показали WordPress та Joomla, оскільки їх вбудована система плагінів разом з вдалою архітектурою дозволяє конструювати проекти широкої функціональної складності. Також під час розгляду інснуючіх рішень було виявленно головний недолік сучасних CMS - використання конкретної СУБД для сховища данних. Такий підхід не є гнучким і система керування контенотом, запропонаванна в данній дипломній роботі намагається вирішити цю проблему.

За функціоналом редагування та фільтрування найкащою виявилась Shopify, тому розроблена в данному дипломному проекті система керування контентом буде орієнтуватися та доповнювати функціонал Shopify.

РОЗДІЛ 2.ОПИС ІНСТРУМЕНТІВ

2.1 Системи управління базами даних

У попередньому розділі було розглянуто приклади сучасних CMS систем, проте всі вони мають спільний недолік — ці CMS системи зав'язані до використання однієй системи управління базами данних (СУБД), а саме: MySQL. СУБД [2] - це загальний термін, що до всіх видів абсолютно різних інструментівк. Так як дані можуть бути різного формату та розміру, були створені різні види СУБД. СУБД засновані на моделях баз даних - певних структурах для обробки даних. Кожна СУБД створена для роботи з однією з них з урахуванням особливостей операцій над інформацією. Хоча рішень, що реалізують різні моделі баз даних, дуже багато, періодично деякі з них стають дуже популярними і використовуються на протязі багатьох років. Зараз найпопулярнішою моделлю є реляційна система управління базами даних (РСУБД).

Запропонованна у цьому проекті CMS система не буде зав'язана до використання конкретної СУБД, її особливість у тому що вона універсальна. Отже, розглянемо які бувають СУБД та моделі данних.

2.1.1 Реляційні СУБД

Реляційна база даних представляє собою набір даних з зумовленими зв'язками між ними. Ці дані організовані у вигляді певної кількості таблиць. У таблицях зберігається інформація про об'єкти, представлених в базі даних. У кожному стовпчику таблиці зберігається певне значення, в кожному осередку - значення атрибута. Кожен рядок таблиці є набором пов'язаних значень, що відносяться до одного об'єкту або сутності. Кожен рядок в таблиці може бути позначена унікальним ідентифікатором, званим первинним ключем, а рядки з декількох таблиць можуть бути пов'язані з допомогою зовнішніх ключів. До цих даних можна отримати доступ багатьма способами, і при цьому реорганізовувати таблиці БД не потрібно. QL (Structured Query Language) - основний інтерфейс роботи з реляційними базами даних. SQL став

стандартом реляційними базами даних. SQL став стандартом Національного інституту стандартів США (ANSI) в 1986 році. Стандарт ANSI SQL підтримується всіма популярними ядрами реляційних БД. Деякі з ядер також включають розширення стандарту ANSI SQL, що підтримують специфічний для цих ядер функціонал. SQL використовується для додавання, оновлення та видалення рядків даних, вилучення наборів даних для обробки транзакцій і аналітичних програм, а також для управління всіма аспектами роботи бази даних.

SQL СУБД мають наступні переваги:

- Надання доступ до даних у реляційних системах управління базами даних.
- Дозволяють користувачам описувати дані
- Дозволяють користувачам визначати дані в базі даних та маніпулювати ними.
- Вбудовуються в інші мови за допомогою модулів SQL, бібліотек та попередніх компіляторів.
- Надають можливість користувачам створювати та скидати бази даних та таблиці.
- Дозволяють користувачам встановлювати дозволи на таблиці, процедури та представлення даних

До найбільш популярних реляційних СУБД відносять:

- SQLite: дуже потужна вбудована СУБД
- MySQL: найпопулярніша СУБД
- PostgreSQL: найпросунітіша і гнучка СУБД

2.1.2 Нереляційні СУБД

Нереляційні СУБД - термін, що позначає ряд підходів, спрямованих на реалізацію систем управління базами даних, що мають суттєві відмінності від моделей, що використовуються в традиційних реляційних СУБД з доступом до даних засобами мови SQL. Застосовується до баз даних, в яких робиться спроба

вирішити проблеми масштабованості та доступності за рахунок атомарності та узгодженості даних. Нереляційні СУБД (NoSQL) спеціально створені для певних моделей даних і мають гнучкими схемами, що дозволяє розробляти сучасні програми. Бази даних NoSQL добре підходять для багатьох сучасних додатків, наприклад мобільних, інтернет-додатків, коли потрібні гнучкі масштабовані бази даних з високою продуктивністю і широкими функціональнимиможливостями, здатні забезпечувати максимальну зручність використання та мають наступні переваги:

- Гнучкість. Бази даних NoSQL, як правило, забезпечують гнучкі схеми, що дають можливість більш швидкого та ітеративного розвитку. Гнучка модель даних робить бази даних NoSQL ідеальними для напівструктурованих та неструктурованих даних.
- Масштабованість. Бази даних NoSQL розроблені для масштабування за допомогою розподілених кластерів обладнання, а не для збільшення масштабів шляхом додавання дорогих і надійних серверів.
- Висока продуктивність. База даних NoSQL оптимізована для конкретних моделей даних та моделей доступу, що дають більш високу продуктивність, ніж намагаються досягти подібної функціональності з реляційними базами даних.
- Широкі функціональні можливості. Бази даних NoSQL надають високофункціональні API та типи даних, призначені для кожної відповідної моделі даних.

Серед найпопулярніших нереляційних СУБД відзначимо:

- MongoDB
- Amazon DynamoDB
- Redis

2.2 Інструменти розробки

Як було зазначено раніше в проекті продемонстровано розробку відповідного програмного забезпечення на основі новітньої технології – React.js та мові програмування JavaScript, у якості СУБД для запропонованої системи керування контентом може бути використана будь-яка SQL або NOSQL база данних, крос-платформний запуск з використанням Docker. Отже, розглянемо детальніше ці технології.

2.2.1 Мова програмування JavaScript

JavaScript [12] - мова програмування, яка відповідає специфікації ECMAScript. Поряд із HTML та CSS, JavaScript є однією з основних технологій веб розробки. JavaScript включає інтерактивні веб-сторінки і є невід'ємною частиною веб-додатків. Переважна більшість веб-сайтів використовують його для поведінки на стороні клієнта, а всі основні веб-браузери мають спеціальний механізм JavaScript для його виконання. Як мова для багато парадигми, JavaScript підтримує керовані подіями, функціональні та імперативні стилі програмування. Він має інтерфейси прикладного програмування (API) для роботи з текстом, датами, регулярними виразами, стандартними структурами даних та Модель об'єкта документа (DOM). Однак сама мова не включає жодного вводу / виводу (вводу / виводу), такого як мережеві засоби, сховища чи графічні засоби, оскільки хост-середовище (як правило, веб-браузер) надає ці API.

2.2.2 Бібліотека React.js

React.js [7] - це бібліотека JavaScript для створення інтерфейсів користувача. React може використовуватись для розробки односторінкових і мобільних додатків. Ціль - представити високу швидкість, простоту і масштабованість. У своїх бібліотеках для розробки інтерфейсів React часто використовується з іншими бібліотеками, такими як Redux та GraphQL.Далі розглянемо приклад найпростішої програми на React.js (рис 2.1):

```
function HelloMessage({ name }) {
  return <div>Hello {name}</div>;
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('container')
);
```

Рис 2.1 Приклад найпростішої програми на React.js

Цей приклад відобразить "Hello Taylor" в контейнер на сторінці. React використовувалє синтаксис, схожий на HTML - JSX. JSX - це спеціальний синтаксис до JavaScript. Рекомендується використовувати цю технологію разом з React, щоб описати, як для опису інтерфейсу користувача. JSX не є обов'язковим для React, але він надає кодц більш читабельний вигляд, і під час написання він виглядає як написання HTML.

2.2.3 Програмне забеспечення Docker

Docker - програмне забезпечення для автоматизації розгортування та управління, що додають у віртуальну систему операційної системи можливість «упаковувати» додаток до всіх його оточенням і залежним у контейнері. Стандарт в індустрії на сьогоднішній день - це використовувати віртуальні машини для використання запропонованих пропозицій. Віртуальні машини використовують додатки всередині гостевої операційної системи, яка працює на віртуальній операційній системі сервера. Віртуальні машини відмінно підходять для повної ізоляції процесора для додатків: майже ніяких проблем, основної операційної системи не може бути використана на м'яких гостьових ОС, і на робочих місцях.

Контейнери використовують інший підхід: вони пропонують схожий з віртуальними машинами рівень ізоляції, але завдяки справжньому застосуванню знижують низькорозвиваючі механізми, основні операційні системи роблять це з урахуванням менших навантажень. Основна проблема з тим, що VM - це додаткова ОС, яка виконує роботу хостової, а також додаткові гігабайти для

проектування. Часто сервер користується VM, і тим самим використовує більше місця. Ще один надійний недолік віртуальної машини - повільне завантаження. Docker розширює всі перераховані проблеми, розділяючи ядро ОС між усіма контейнерами, які працюють як окремі хостові ОС (рис 2.2):

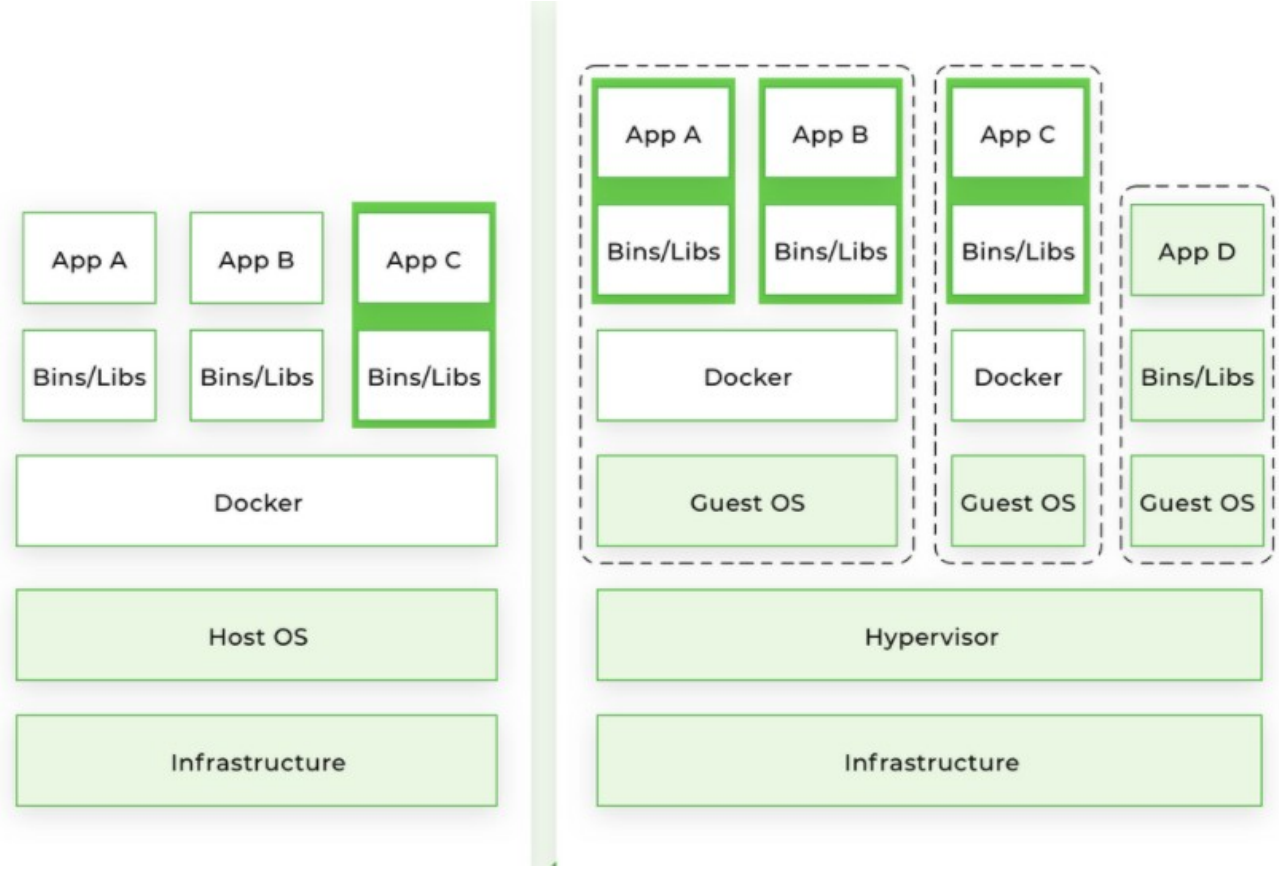


Рис 2.2 Приклад роботи контейнерів

За допомогою Docker можливо створити, запустити, зупинити, перемістити або видалити контейнер. Можна підключити контейнер до однієї або декількох мереж, приєднати до нього сховище або навіть створити нове зображення залежно від його поточного стану. За замовчуванням контейнер відносно добре відокремлений від інших контейнерів та його хост-машини. Можливо керувати тим, наскільки ізольованою є мережа, сховище чи інші базові підсистеми контейнера від інших контейнерів або від хост-машини. Контейнер визначається будь-якими параметрами конфігурації, які надаються йому під час створення або запуску. Коли контейнер видалений, будь-які зміни його стану, які не зберігаються в постійному сховищі, зникають.

2.3 Розробка системи

Надалі буде описана розробка системи керування контентом, яка в свою чергу повинна буди спроможна виконувати наступні маніпуляції з данними:

- Відображати данні з певного сховища данних
- Сортувати та фільтрувати данні
- Редагувати данні

Перед описом розробки системи дуже важлив познайомитись з файловою структурою системи (рис 2.3), тому розглянемо її:

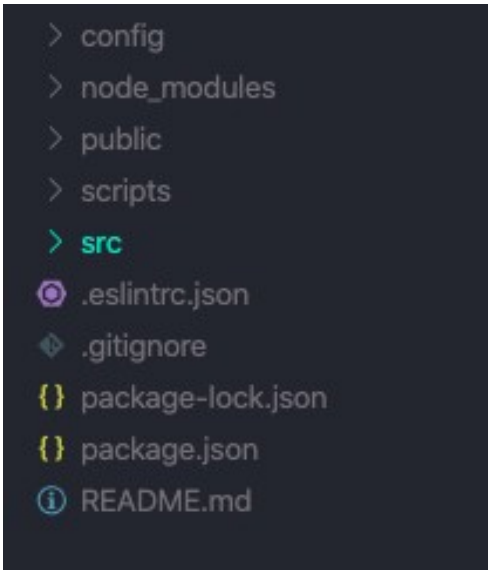


Рис 2.3 Файлова структура системи

Головна частина вихідного коду розташована у папці «src», інші папки та файли:

1) папка «config» - головна конфігурація запуску системи
2) папка «node_modules» - сторонні модулі системи
3) папка «public» - скомпільовані компоненти системи
4) папка «scripts» - скрипти запуску та білду системи
5) README.md - файл з описом системи
6) package.json та package-lock.json — файли з описом залежностей
7) .eslintrc.json — файл з правилами написання вихідного коду

2.3.1 Розробка веб компонентів системи

Веб компоненти являють собою відокремленя логіку відображення данних та можуть бути використані повторно в рамках системи, тобто головна їх особливість — універсальність та можлівисть повторного використання. Усі вебкомпоненти будуть розташовані а папці «shared» (рис 2.4):

Рис 2.4 Контент папки «shared»

Папка «shared» містить п'ять головних компонентів, які можуть бути використані повторно вздовж усієї системи:

1) «common-paginator» (рис 2.5) - компонент, який надає можливість для будь якої таблиці відображати данні у вигляді сторінкового інтерфейсу з двома головним параметрами: номер сторінки та обсяг сторінки данних:

Рис 2.5 Компонент «common-paginator»

Найважливішими у данному компоненті є параметер «page», який відповідає за номер сторінки, та параметер «pagesize», який відповідає за обсяг данних на сторінкі.

2) «common-spinner» (рис 2.6) - компонент, який відображається у системі коли данні з серверу ще не загрузилися. Данний компонент є дуже важливим з точки зору користувача, оскільки нотифікує його що данні в процесі загрузки:

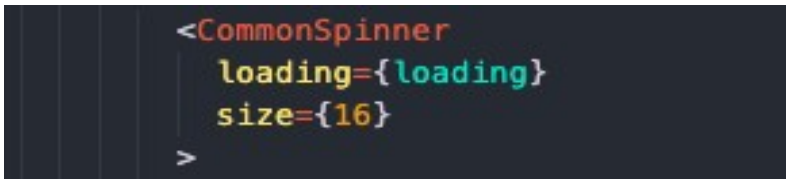


Рис 2.6 Компонент «common-spinner»

Компонент приймає два параметри: «loading», який вказує чи відображати компонет та «size», який є опціональним і вказує на розмір спінеру.

3) «common-table» (рис 2.7) - контейнером відображення у вигляді таблиці:

Рис 2.7 Компонент «common-table»

Головними параметрами компоненту є параметер «headers» - масив заголовків стовпчиків таблиці, та параметер «rows» - масив данних який буде відображено в таблиці. Усі інші параметри є опціональними.

4) «error-message» (рис 2.8) — компонент є важливим с точки зору користувача, який повідомляє його, що данні не можуть буди відображені, якщо відбулася помилка на стороні серверу:

```
<ErrorMessage error={error}></ErrorMessage>
```

Рис 2.8 Компонент «error-message»

Компонент є тривіальним і дуже простим, приймає лише один параметер, а саме: «error» - об'єкт повідомлення про помилку.

5) «nested-list» (рис 2.9) — базовий компонент для відображення данних у вигляді випадаючого списку:

```
<NestedList
  items={items}
  onItemClick={onItemClick}
  error={error}
/>
```

Рис 2.9 Компонент «nested-list»

Компонент має головний параметер - «items», який є масивом єлементів списку, усі інші параметри є опціональними та можеть бути опущені.

Вище були описані головні компоненти систем, які можуть бути динамічно перевикористані в рамках системи. Надалі розглянем найголовніший компонент системи та його взаємодію з іншими компонентами.

Найголовнішим компонентом системи є компонент «core-table», який знаходиться у папці «core» → «components» → «core-table» (рис 2.10):

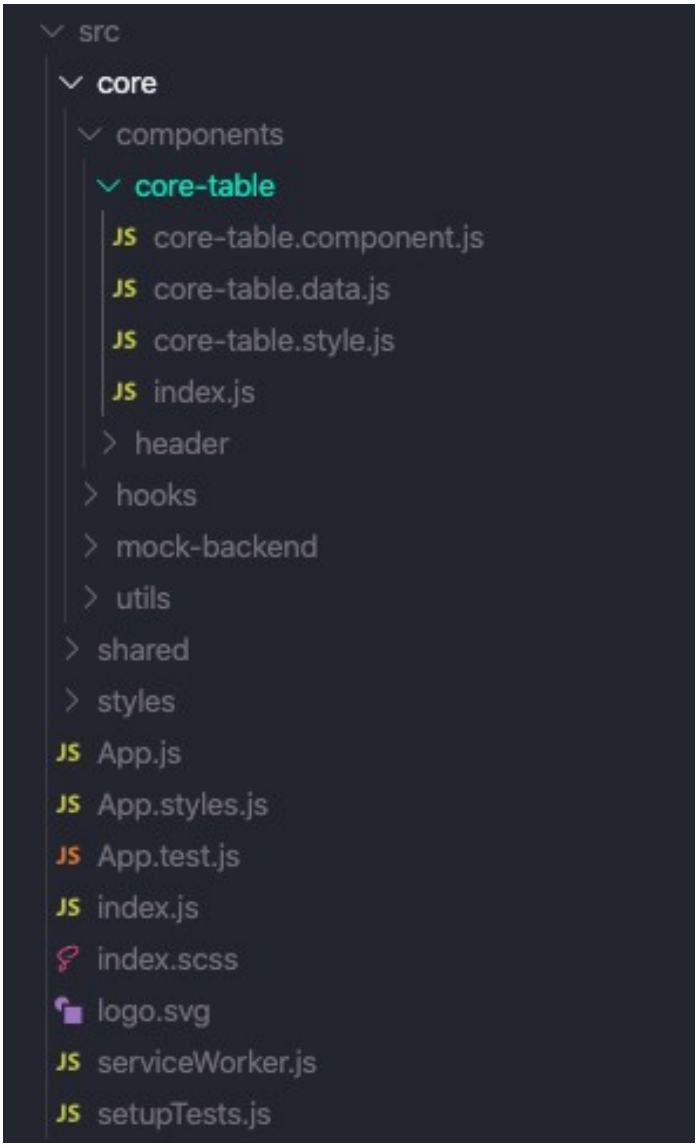


Рис 2.10 Розташування компоненту «core-table»

Компонент «core-table» (рис 2.11) є найголовнішим компонентом систем, з яким користувач починає відразу ж взиємодіяти після запуску системи керування контентом. Як видно з файлової структури компоненту, він містить файл «core-table.data.js», в якому зосереджена головна логіка запитів для маніпулювання з данними. Файл «core-table.style.js» містить стильову конфігурацію компонента та є дуже важливим з точки зору користувача. Також, данний компонент інкапсулює відображення данних у вигляді таблиці, здійснює запити для отримання данних і належного їх відображення та лінкує інші компоненти у своєму представленні:

Рис 2.11 Компонент «core-table»

Як видно з вихідного коду компоненту «core-table» данний компонент інкапсулює чотири компоненти:

- «common-spinner» - компонент використовується щоб нотифікувати користувача шо данні в процесі загрузки
- «error-message» - компонент використовується щоб відображати помилки серверу
- «common-table» - компонент для відображення данних користувача у

вигляді таблиці

- «common-paginator» - компонент для відображення данних у вигляді сторінкового інтерфейсу

Також данний компонент має внутрішний стан (рис 2.12) та зберігає у ньому необхідні змінні для усіх вище зазначених компонентів:

```
const [headers, setHeaders]       = useState([]);
const [rows, setRows]             = useState([]);
const [count, setCount]           = useState(0);
const [notFilter, setNotFilter]   = useState({});
const [loading, setLoading]       = useState(true);
const [error, setError]           = useState(null);
```

Рис 2.12 Внутрішній стан компоненту «core-table»

Як видно з внутрішнього стану компоненту він містить

- «headers» - масив заголовків таблиці
- «rows» - масив данних для відображення
- «count» - кількість рядків таблиці
- «notFilter» - данні для фільтруванн
- «loading» - індентифікатор загрузки данних
- «error» - об'єкт помилки серверу

Висновок

Було розглянено структуру системи керування контентом та описано взаємодію компонентів системи. Як ми побачили найголовнішим компонентом системи є компонент - «core-table», який інкапсулює у собі інші компоненти та є першочерговим контейнером представлення даних.

Також система містить п'ять головних компонентів, які строворені для повторного використання вздовж усієї системи. Компоненти мають низку важливих параметрів у своєму представленні, які також було детально описано та вказано на доцільність використання.

2.3.2 Архітектурний стиль REST веб компонентів системи

REST - архітектурний стиль взаємодії компонентів розподіленого додатка в мережі. REST є узгоджений набір обмежень, що враховуються при проектуванні розподіленої гіпермедіа-системи. У певних випадках (інтернет-магазини, пошукові системи, інші системи, засновані на даних) це призводить до підвищення продуктивності і спрощення архітектури.

Системи, що слідують парадигмі REST, не мають стану, тобто сервер не повинен нічого знати про те, в якому стані знаходиться клієнт, і навпаки. Таким чином і сервер, і клієнт можуть зрозуміти будь-яке отримане повідомлення, навіть не знаючи про попередні повідомленя. Це обмеження здійснюється через використання ресурсів, а не команд. Ресурси - це іменники Інтернету - вони описують будь-який об’єкт, документ чи річ, які, можливо, знадобляться для зберігання чи надсилання іншим службам. Використовуючи архітектурний стиль REST, різні клієнти потрапляють на одні і ті ж кінцеві точки REST, виконують ті самі дії та отримують однакові відповіді.

REST архітектура у своєму стилі означає що в системі буде набір «розумних» endpoint — тів, які мають детерміновану поведінку, тобто приймають первні параметри та повертають в залежності від цих параметрив данні. REST запити восновному поділяються на GET запити та на POST запити. GET відповідають за отримання данних тільки для читання і є readonly, в той час як POST запити є запитами модифікації данних, та можуть повертати змінені данні.

Система керування контентром дипломного проекту може здійснювати наступні GET для відображення данних:

1) GET /api/collections (рис 2.13) — запит для отримання данних випадаючого меню для відображення певних коллекцій з бази данних. Данний GET запит є дуже важливим з точки зору системи, оскільки без його належної роботи неможливо буде відобразити жодні данні. Як видно з запиту він не має жодного параметру.

```
useEffect(() => {
  axios.get("api/collections", {})
    .then(response => {
      return response.data.elements;
    })
    .then(response => {
      setItems(response);
      setError(null);
    }).catch(err => {
      setError({
        errorCode: err.response.status,
        errorMessage: err.message
      });
    });
}, []);
```

Рис 2.13 Приклад запиту GET /api/collections

2) GET /api/{database}/{collection}?pagesize=10&page=1 (рис 2.14) — запит для отримання данних з певної бази данних та колекції. Даний запит є необхідним для системи оскільки він слугує для відображення данних у вигляді сторінкового інтерфейсу. Запит має наступні параметри:

- database — параметер є необхідним та відповідає за базу данних до якої йде підключення
- collection - параметер є необхідним та відповідає за колекцію бази данних до якої йде підключення
- pagesize — параметер є опціональним (якщо не вказаний то буде використано значення «10») і відповідає за кількість рядків таблиці для відображення
- page - параметер є опціональним (якщо не вказаний то буде використано значення «1») і відповідає за номер сторінки таблиці для відображення

```
export const fetchTableData = (endpoint, page, pagesize, keys, filter) => {
  return () => {
    return axios.get(endpoint, {
      params: {
        pagesize,
        page,
        keys,
        filter
      }
    }).then(res => res.data);
  }
};
```

Рис 2.14 Приклад запиту GET /api/{database}/{collection}?pagesize=10&page=1

3) GET /api/{database}/{collection}?pagesize=10&page=1&filter={"not": []} (рис 2.15) - запит для отримання фільтрованних данних з певної бази данних та колекції. Даний запит слугує для відображення фільтрованих данних системи у вигляді сторінкового інтерфейсу. Запит має наступні параметри:

- database — параметер є необхідним та відповідає за базу даних до якої йде підключення
- collection - параметер є необхідним та відповідає за колекцію бази даних до якої йде підключення
- pagesize — параметер є опціональним (якщо не вказаний то буде використано значення «10») і відповідає за кількість рядків таблиці для відображення
- page - параметер є опціональним (якщо не вказаний то буде використано значення «1») і відповідає за номер сторінки таблиці для відображення
- filter — параметер є необхідним та представленний у вигляді масива певних полів, за якими йде беспосереднь фільтрація

```
useEffect(() => {
  useFetch(
    fetchTableData(
      dataPath,
      page + 1,
      pagesize,
      {},
      { "$not": notFilter }
    ),
    setLoading,
    setError
  )
    .then(res => {
      setHeaders(Object.keys(res.elements[0] || []));
      setRows(res.elements);
      setCount(res.totalElements);
    })
    .catch(e => e);
}, [dataPath, page, pagesize]);
```

Рис 2.15 Приклад запиту GET /api/{database}/{collection}?pagesize=10&page=1&filter={"not": []}

Також варто відзначити що кожен з відповідей даних GET запитів має мати один і той же інтерфейс даних (рис 2.16), а саме:

```
▼{totalPages: 1, totalElements: 40, elements: Array(40), page: 1, pagesize: 1000}
  ▶elements: (40) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}
   page: 1
   pagesize: 1000
   totalElements: 40
   totalPages: 1
  ▶__proto__: Object
```

Рис 2.16 Інтерфейс GET запитів

Як видно з інтерфейсу, відповідь з GET запиту містить спільні поля:

- elements – безпосередньо масив даних який буде повернено GET запитом
- page – поточний номер сторінки сторінкового інтерфейсу даних
- pagesize - поточний обсяг сторінки сторінкового інтерфейсу даних

- totalElements – загальна кількість елементів у сховищі даних
- totalPages – загальна кількість сторінок сторінкового інтерфейсу сховища даних

На данний момент система містить один POST запит, який відповідний за редагування даних (рис 2.17):

Рис 2.16 POST запит для редагування даних

Даний POST запит на редагування даних приймає наступні параметри:

- endpoint — шлях до сховища даних, представлений у вигляді {database}/{collection}/{id}, де database — база даних, до якої відбувається підключення; collection - колекція бази даних, до якої відбувається підключення; id — унікальний індентифікатор даних, які підлягають редагуванню
- data – інтерфейс даних для редагування (рис 2.17):

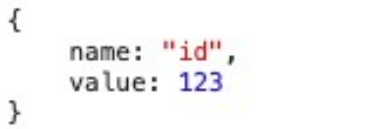


Рис 2.17 Інтерфейс даних для редагування

Данний інтерфейс представляє собою лише два поля:

- name – назва поля для редагування
- value – нове значення поля

2.3.3 Режими роботи системи

Під час роботи система підтримує два режима роботи:

- REAL - режим у якому данні для системи беруться з серверу
- MOCK - режим у якому у якому відображаються фейкові данні

Під час роботи системи можна перемикатися з одного режиму у інший, для цього необхідно лиш виконати наступні команди у консолі браузера:

1) window.setMockMode() - для MOCK режиму

2) window.setRealMode() - для REAL режиму

2.3.4 Классифікація CMS системи

До сих пір не розроблено достатньо чіткої класифікації [1] систем управління контентом. Це відбувається тому, що ринок CMS достатньо молодий, і розробники такого роду програмних продуктів в значній мірі роз'єднані. Складно розділити їх на будь-які групи ще й тому, що всі вони досить сильно відрізняються один від одного. Тому будь-яку класифікацію можна назвати в достатній мірі умовною. В ході досліджень було виділено 3 критерії класифікації CMS:

1) за рівнем складності

2) за способом роботи

3) за способом поширення

4) за областями застосування

Для класифікації за рівнями складності [3] можна виділити наступні характеристики:

- наявність тих чи інших функцій і модулів, зрозумілість і доступність користувачеві;

- можливість функціонування системи на різних платформах, сумісність з базами даних, можливість підключення додаткових модулів;

- технологічність - використання технологій, що дозволяють підвищити надійність і швидкодію системи;

В свою чергу для класифікації за способом роботи можна виділити

наступні види:

1. Генерація сторінок за запитом. Системи такого типу працюють на основі зв'язки «Модуль редагування → База даних → Модуль уявлення». Модуль уявлення генерує сторінку з вмістом при запиті на нього, на основі інформації з бази даних. Інформація в базі даних змінюється за допомогою модуля редагування. Сторінки заново створюються сервером при кожному запиті, що в свою чергу створює додаткове навантаження на системні ресурси. Навантаження може бути багаторазово знижена при використанні коштів кешування, які є в сучасних web-серверах.

2. Генерація сторінок при редагуванні. Системи цього типу суть програми для редагування сторінок, які при внесенні змін до змісту сайту створюють набір статичних сторінок. При такому способі в жертву приноситься інтерактивність між відвідувачем і вмістом сайту.

3. Змішаний тип. Як зрозуміло з назви, поєднує в собі переваги перших двох. Може бути реалізований шляхом кешування - модуль уявлення генерує сторінку один раз, надалі вона в кілька разів швидше завантажується з кешу. Кеш може оновлюватися як автоматично, після закінчення деякого терміну часу або при внесенні змін до певні розділи сайту, так і вручну по команді адміністратора. Інший підхід - збереження певних інформаційних блоків на етапі редагування сайту і збірка сторінки з цих блоків при запиті відповідної сторінки користувачем. Отже, до переваг CMS слід віднести наступні:

1. Швидке і ефективне управління інформацією. Можливість підключення до наповнення сайту не тільки спеціального web-майстра або редактора, але і всіх співробітників, що володіють тією чи іншою інформацією.

2. Зменшення термінів і вартості розробки тих чи інших функцій і надання додаткових сервісів.

3. Підвищення якості розробки та зміни сайту. Так як всі ці розробки вже пройшли неодноразове тестування і давно використовуються.

4. Зниження вартості подальших змін, за рахунок поділу даних і їх

уявлення.

5. Зниження вартості підтримки або взагалі повна її відсутність. Так як власник сайту може сам змінювати щось, без участі web-майстра або розробників.

Для класифікації за областями застосування [4] можна виділити такі типи:

1. Магазини. До магазинів відноситься будь-який сайт, з якого можна замовити будь-який товар. В даному випадку в визначення «товару» може входити абсолютно все, включаючи час доступу в Інтернет, хвилини стільникового зв'язку. Абсолютна більшість інтернет-магазинів є нелегальними. CMS, що дозволяють створити віртуальний магазин: MyMarket, osc2nuke, osCommerce, Zen Cart.

2. Форуми - це інструмент для спілкування на сайті. Повідомлення у форумі в чомусь схожі на поштові - кожне з них має автора, тему і зміст. Але для того, щоб відправити повідомлення в форум, не потрібна ніяка додаткова програма - потрібно просто заповнити відповідну форму на сайті. Принципова властивість форуму полягає в тому, що повідомлення в ньому об'єднані в треди (від англ. Thread - «нитка»). Коли відповідаєте в форумі на чиєсь повідомлення, ваш відповідь буде «прив'язаний» до вихідного повідомлення. До форумів, гідним уваги, можна зарахувати FUDforum, openBB, Phorum, phpBB, PunBB.

3. Портали. Використовуються для інформаційних ресурсів, основною метою ставлять максимальне спрощення публікації статей і новин. Можуть включати в себе перераховані нижче типи CMS як самостійні модулі. Найбільш відомі представники даного класу: AngelineCMS, Bes-cms, CoolPHP, CPG-Nuke, вебZE, Xaraya, xNuke, XOOPS і ін.

4. Навчання (e-Learning) - дистанційна форма навчання з використанням Інтернету. Онлайнова форма навчання вже не один рік є «маяком», на який орієнтуються освітні системи різних країн світу. головним стратегічним напрямком є швидке оновлення знань і ефективне використання інформації.

Таких систем небагато: ATutor, Claroline, LogiCampus, Moodle, Segue, Site @ School, eLearning 3000.

5. Адміністраторська панель хостингу. До цього класу належать такі продукти, як Direct Admin і Control Panel. Чимало хостинг-провайдерів намагаються написати панель управління для користувача хостингу своїми силами, проте жодне подібне рішення, наскільки відомо, так і не змогло за можливостями і ергономіці хоч трохи наблизитися до вищезазначених систем.

До основних недоліків CMS систем слід віднести наступні.

1. Шаблоновий дизайн і розташування елементів сайту. більшість популярних CMS систем не дозволяють створювати сайти з індивідуальним дизайном.

2. Неможливість додавання власних динамічних блоків. Ця риса властива розрахованих на багато користувачів CMS систем, в які додавати динамічні блоки може тільки адміністрація сайту.

3. Неуніверсальність CMS систем. Через обмеженого набору числа динамічних блоків і неможливості простого створення власних динамічних блоків CMS системи не здатні створювати сайти будь-якої складності.

За способом поширення системи [5] керування контентом бувають вільно-розповсюджуваними і платними. Разом з останніми в переважній більшості випадків клієнти отримують супровід і підтримку. Платні системи можна розділити на три цінові категорії:

1. Найбільш дешеві, які вироблені одним web-розробником або групою. Такі системи намагаються зробити якомога більше універсальними, щоб продати якомога більшій кількості потенційних покупців (як правило, інтернет-представництв дрібних компаній).

2. Системи середнього цінового діапазону, які розробляються під конкретного замовника.

3. Системи, створені гігантами розробки - Google і ін.

ВИСНОВКИ ДО РОЗДІЛУ 2

В данному розділі було розглянуто та вказано на доцільність використання засобів розробки програмного забеспечення, зокрема мова програмування JavaScript, яка надає велику кількість можливостей для вирішення найрізноманітніших завдань, JavaScript бібліотеку React.js для побудови клієнтської частини, та систему контернізації Docker, який в свою чергу надає можливість кросплатформеного запуску.

Було описано головні компоненти системи та їх взаємодію. Серед головних п'яти компонентів які можуть бути повторно використані вздовж усієї системи відзначено: «common-spinner», «error-message», «common-paginator» «nested-list», «common-table». Найголовнішим компонентом системи є компонент - «core-table», який в свою чергу є контейнером відображення даних і інкапсулює у собі внутрішній стан інших компонентів. Усі компоненти в системі взаємодіють за допомогою архітектурного стилю компонентів розподіленого додатка в мережі — REST. Система має три основні GET запити для отриманная т відображення даних у виді таблиці та один POST запит для редагування даних. Всі GET та POST запити у системі мають один і той же інтерфейс відповіді на запит, а саме:

- elements – масив даних
- page – поточний номер сторінки
- pagesize - поточний обсяг сторінки
- totalElements – загальна кількість елементів
- totalPages – загальна кількість сторінок

Також варто уваги те що система може працювати у двух режимах, а саме: MOCK та REAL, що надає можливість системі працювати навіть у оффлайн моді. Дана особливість може використовуватись у демонстраційних цілях можливостей системи керування контентом.

Також, було проведено классифікацію CMS систем та вказано на недоліки

сучасних CMS. Виходячи з класифікації можна зазначити що розроблена у данному дипломному проекті система керування контентом є системою “генерація сторінок за запитом” та являється універсальною CMS системою, оскільки має просунуті засоби керування контентом.

РОЗДІЛ 3. ОПИС СИСТЕМИ ТА ІНСТРУКЦІЯ КОРИСТУВАЧА

3.1 Інструкція користувача

Щоб почати роботу – запускаємо програму. Відкривається стартове вікно системи керування контентом, навігаційна панель користувача включає всі джерела даних (рис. 3.1), будь-то реляційна, нереляційна СУБД, або навіть файл. Також у стартовому вінкі зліва буде видно список усіх коллекцій конкретної бази даних, данний список реалізовано у виді випадаючого. По центру знаходиться головна таблиця (рис 3.2) системи керування контентом, також знизу таблиці знаходиться панель для навігації по сторінках таблиці.

Далі користувач може обрати конкретне джерело даних на навігаційній панелі, лише клікнувши мишею на бажаний пункт, таким чином будуть завантажені все схеми та коллекції (рис. 3.3) з відповідного джерела даних. Також можливо переключатися з одної коллекції даних на іншу просто клікнувши на бажану коллекцію.

Данні у запропонованній системі керування контентом представлені у виді сторінок, оскільки певна колекція може містити забагато даних і відображення їх усіх в конкретний проміжок часу не є ефективним. Для цього у нижньому правому кутку системи розташованні опції для вказання конретної розмірності сторінки даних (рис. 3.4). Користувач може обрати зручну кількість для себе і система автоматично перегрупує набір даних.

Редагування даних реалізовано через дуже зручний інтерфейс, достатнь лише поставити курсор миші у базаже місце і виправити данні. Дуже важливою характеристикою системи керування контентом є можливість фультрувати данні, у запропонованній системі ця можлівість наявна у кожному стопчику (рис. 3.5). Випадаючий список для фільтрування має наступні одночасно важливі та зручні фунції для користувача:

- можливість обрати усі пункти фільтру. Для цього необхідно лише клікнути на «Select All» чекбокс

- можливість пошуку по всім значенням у таблиці. Для цього необхідно необхідно лише ввести значення у текстове поле вводу і зняти курсор миші
- можливість обрати окреме поле у списку фільтрів. Для цього необхідно лише клікнути на бажане поле курсором миші

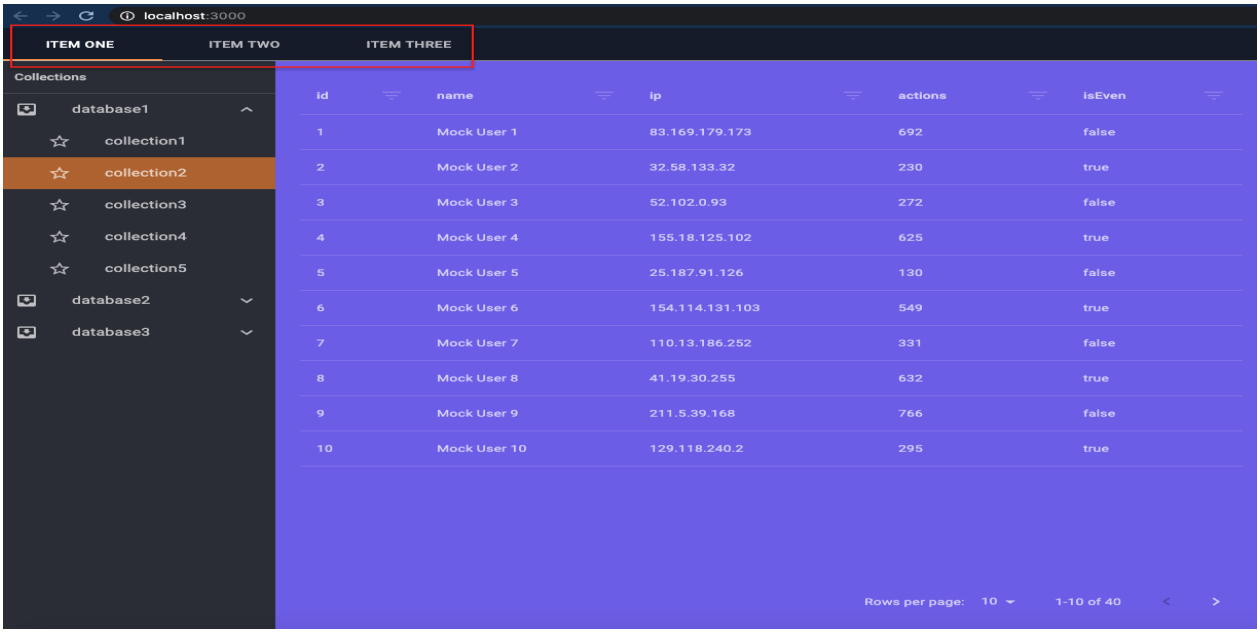


Рис. 3.1 Навігаційна панель користувача

Рис 3.2 Головна таблиця системи

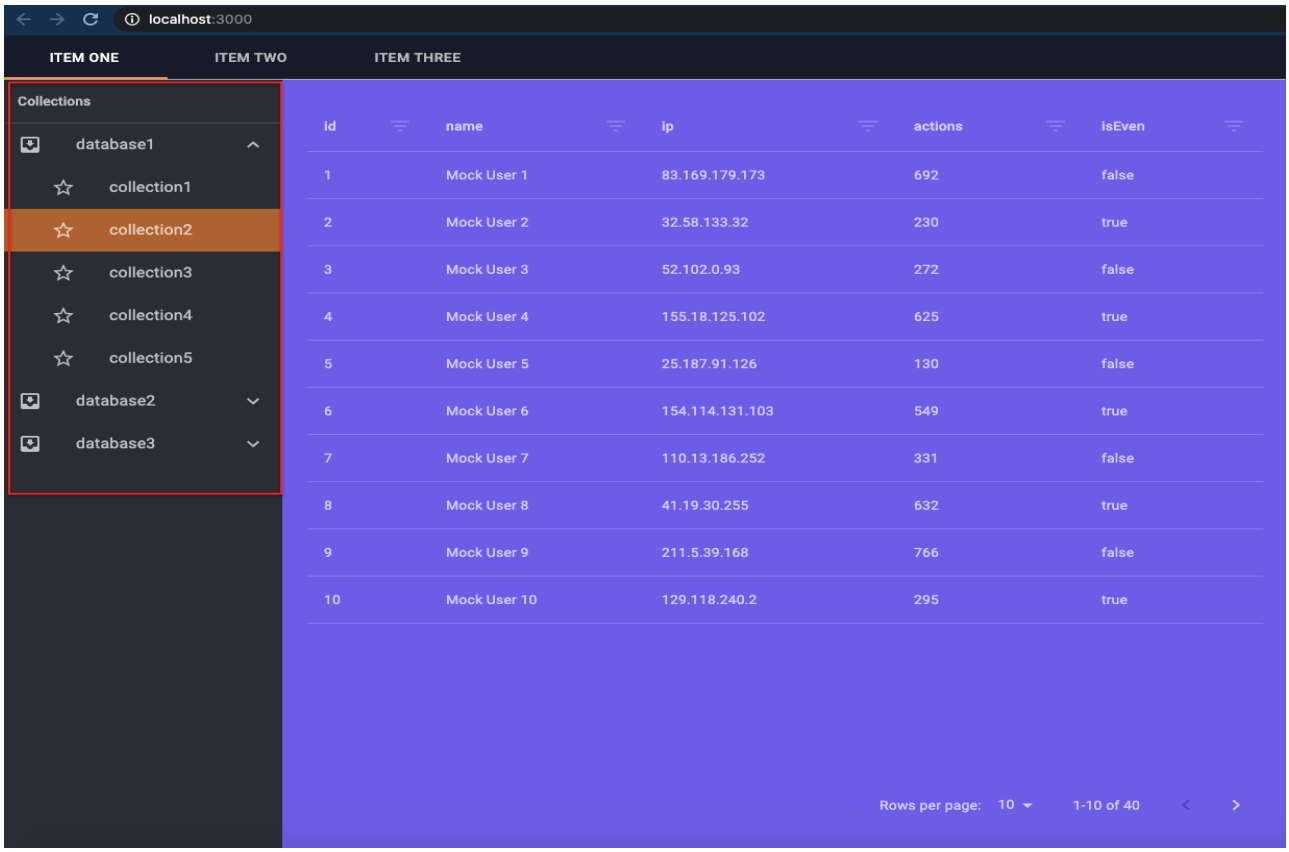


Рис. 3.3 Всі схеми та колекції

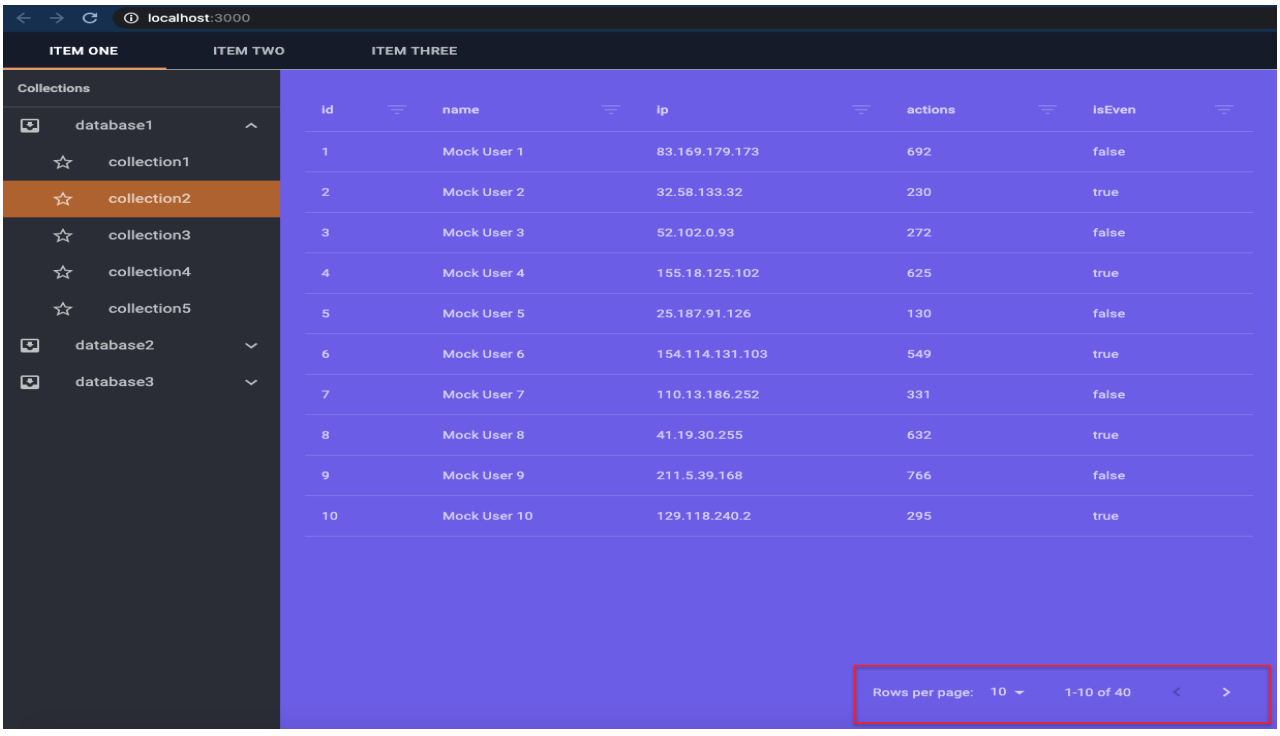


Рис. 3.4 Розмірності сторінки даних

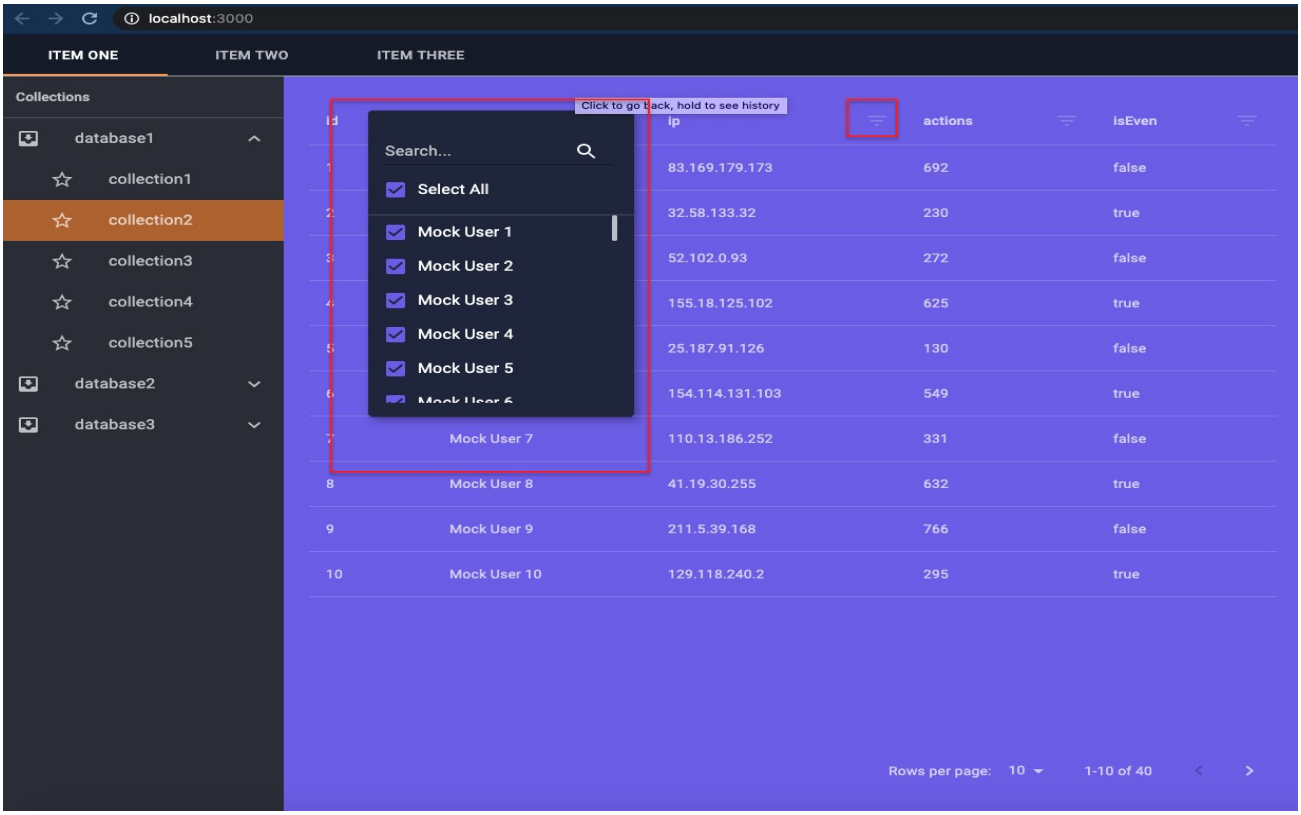


Рис 3.5 Можливість фультрувати данні

3.2 Тестування системи

Система керування контентом була протестована двома шляхами: юніт тестуванням та ручним тестуванням.

Юніт тестування (рис 3.6) - також відоме як модульне тестування, використовується під час розробки програмного забеспечення та дозволяє перевірити коректність роботи певних модулів. Основна ідея данного методу тестування - це написати низку тестів для функцій або методів, що дозволяє перевірити чи не призвели нові зміни у вихідному коді програми до регресії. Юніт - це найменша перевірена частина будь-якого програмного забезпечення. Зазвичай він має один або кілька входів і, як правило, один вихід. У процедурному програмуванні юнітом може бути індивідуальною програмою, функцією, процедурою тощо. У об'єктно-орієнтованому програмуванні юніт - це метод, який може належати до базового / надкласного, абстрактного класу або класу похідних / дочірніх. (Деякі ставляться до модуля програми як до одиниці. Це слід відмовити, оскільки, ймовірно, буде багато окремих блоків у цьому

модулі). Рамки тестування блоків, драйвери, заглушки та макетні / підроблені об'єкти використовуються для допомоги в тестуванні одиниць. Розглянемо переваги, які надає юніт тестування:

- Підвищує впевненість у зміні / підтримці коду. Якщо будуть написані хороші юніт тести, і якщо вони виконуються щоразу, коли будь-який код буде змінено, можливо оперативно зафіксувати будь-які дефекти, введені через зміну. Крім того, якщо модулі вже зроблені менш взаємозалежними, щоб зробити можливим тестування одиниць, навмисний вплив змін до будь-якого коду менший.
- Вартість виправлення дефекту, виявленого під час тестування, менша порівняно з вадою, виявленою на більш високих рівнях.
- Легке налагодження. Якщо тест не є вдалим, слід налагоджувати лише останні зміни. При тестуванні на більш високих рівнях зміни, внесені протягом декількох днів / тижнів / місяців, потрібно сканувати.
- Розробка програмного забеспечення значно швидше. Якщо у програмі немає юніт тестування, пішеться вихідний код і виконується те нечітке ручне тестування. Але, якщо є юніт тестування, коли пишеться тест, то пишется і код до нього, та тест відразу ж запускається. Написання тестів вимагає часу, але час компенсується меншою кількістю часу, необхідного для виконання тестів. Не потрібно запускати графічний інтерфейс і тестувати вручну. І, звичайно, юніт тести є більш надійними, ніж ручне тестування. Також, зусилля, необхідні для пошуку та виправлення дефектів, виявлених під час юніт тестування, є значно меншими порівняно з зусиллями, необхідними для виправлення дефектів, виявлених під час тестування системи вручну.

В данному дипломному проекті написана низка юніт тестів, які пожуть бути виконані простою командою - «npm run test». Необхідно лише запустити цю команду і буде отримано результати тестування:

```
/* global test expect */

import React from "react";
import { render } from "@testing-library/react";
import App from "./App";

test("renders learn react link", () => {
  const { getByText } = render(<App />);
  const linkElement = getByText(/collections/i);
  expect(linkElement).toBeInTheDocument();
});
```

Рис 3.6 Приклад юніт тесту

Під час запуску юніт тестів командою «npm run test» будуть виведені логи результатів тестування (рис 3.7), в яких буде видно загальну кількість вдалих та невдалих тестів:

```
Test Suites: 1 passed, 1 total
Tests:       34 passed, 34 total
Snapshots:   0 total
Time:        8.397s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.
```

Рис 3.7 Результати юніт тестування

Як видно з результатів система містить 34 юніт тести, які були виконані вдало, що вказує на те що система працює коректно.

Під час ручного тестування було перевірено належну роботу наступних функцій:

1) відображення списку коллекцій бази даних

2) відображення табличних данних у вигляді сторінкового інтерфейсу

2) відображення табличних данних у вигляді сторінкового інтерфейсу

3) робота пагінації таблиці

4) фільтрування данних у випадаючому списку фільтрів

5) редагування данних

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі була розглянута інструкція користувача. Програма має легкий, зрозумілий та зручний інтерфейс для аналізу та редагування даних. Також було продемостровано головні особлівості запропонованної системи керування контентом, зокрема фільтрацію, групування та редагування.

Було проведено юніт тестування разом з ручним тестуванням, що надало можливість пересвідчитися у коректності роботи системи керування контентом.

ВИСНОВОК

Результатом бакалаврської роботи є система керування контеном, головна перевага якої перед існуючими системами – можливість використовувати будь-яке джерело данних, будь-то реляційна, нереляційна СУБД або навіть файл. Також запропонавана система має значні перевагі у зручності інтерфейсу користуваяа порівняно з її аналогами.

Для написання системи були використанні новітні інструменти та технології такі як React.js та Docker.

На даний момент система може бути доповнена автоматичним тестуванням, розширенням функціоналу, підтримкою на редагування данних повної группи користувачів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Бурніков М.Ю. CMS огляд [Електронний ресурс] / М.Ю. Бурніков, Н.С. Акімов. - Режим доступу: http://cmsobzor.ru.

[2] Документація СУБД [Електронний ресурс] Режим доступу : https://lecturesdb.readthedocs.io/databases/dbms.html

[3] Замеріна В.В. Система управління вмістом [Електронний ресурс] / В.В. Замеріна, Д.Б. Головін. - Режим доступу: http://cmslist.ru/.

[4] Щеглова Е.А. Класифікація CMS [Електронний ресурс] / Е.А. Щеглова, П.К. Павлов. - Режим доступу: http://stroimweb.moy.su/publ/6.

[5] Шелестов Н.Д. Перспективы CMS [Электронный ресурс] / Н.Д. Шелестов – Режим доступу http://whatis.techtarget.com/definition/gci508916,00.html.

[6] Документація GNU [Електронний ресурс]. Режим доступу: https://www.debian.org/releases/stable/mips/ch01s02.ru.html.

[7] Документація React [Електронний ресурс]. Режим доступу: https://github.com/facebook/react .

[8] Документація WordPress [Електронний ресурс]. Режим доступу: https://uk.wordpress.org.

[9] Документація Drupal [Електронний ресурс]. Режим доступу: https://www.drupal.org.

[10] Документація Joomla [Електронний ресурс]. Режим доступу: https://www.joomla.org.

[11] Документація Shopify [Електронний ресурс]. Режим доступу: https://www.shopify.com.

[12] Специфікація JavaScript [Електронний ресурс]. Режим доступу: https://learn.javascript.ru/.

ДОДАТОК А

Система керування контентом

Лістинг частин програми

ІАЛЦ.466521.005 Д1

Аркушів 12

Київ — 2020 року

Лістінг common-paginator.component.js

```
import React from "react";
import PropTypes from "prop-types";
import TablePagination from "@material-ui/core/TablePagination";

import { useStyles } from "./common-paginator.styles";

export const CommonPaginator = ({
      pagesizeOptions,
      count,
      pagesize,
      page,
      handleChangePage,
      handleChangePagesize
}) => {
const classes = useStyles();

      return (
            <TablePagination
                  component="div"
                  count={count}
                  onChangePage={handleChangePage}
                  onChangeRowsPerPage={handleChangePagesize}
                  page={page}
                  rowsPerPage={pagesize}
                  rowsPerPageOptions={pagesizeOptions}
                  classes={{
                  root: classes.root,
                  caption: classes.caption,
                  selectIcon: classes.selectIcon,
                  select: classes.select,
                  actions: classes.actions
            }}>
            </TablePagination>
      );
}

CommonPaginator.propTypes = {
    pagesizeOptions: PropTypes.arrayOf(PropTypes.number).isRequired,
    count: PropTypes.number.isRequired,
    pagesize: PropTypes.number.isRequired,
    page: PropTypes.number.isRequired,
    handleChangePage: PropTypes.func.isRequired,
    handleChangePagesize: PropTypes.func.isRequired
}
```

Лістінг common-spinner.component.js

```
import React from "react";
import PropTypes from "prop-types";
import GridLoader from "react-spinners/GridLoader";

import { warningColors } from "@styles/variables";
import { useStyles } from "./common-spinner.styles";

export const CommonSpinner = ({ loading, size }) => {
      const classes = useStyles();

      return (
            <div
                  className={classes.loadingShade}
                  style={{
                        display: loading ? "flex" : "none"
                  }}
            >
                  <GridLoader
                        color={warningColors.main}
                        loading={loading}
                        size={size}
                  >
                  </GridLoader>
            </div>
      );
}

CommonSpinner.propTypes = {
      loading: PropTypes.bool,
      size: PropTypes.number
}
```

Лістінг common-table.component.js

```
import React from "react";
import PropTypes from "prop-types";
import Table from "@material-ui/core/Table";
import TableBody from "@material-ui/core/TableBody";
import TableCell from "@material-ui/core/TableCell";
import TableContainer from "@material-ui/core/TableContainer";
import TableHead from "@material-ui/core/TableHead";
import TableRow from "@material-ui/core/TableRow";
import Paper from "@material-ui/core/Paper";

import { ColumnFilters } from "./column-filters";
import { useStyles } from "./common-table.styles";

export const CommonTable = ({
      fetchEffect,
      headers,
      rows,
      onFilterSelect,
}) => {
      const classes = useStyles();

      return (
            <TableContainer className={classes.table} component={Paper}>
                  <Table
                        // stickyHeader
                        style={{
                              borderCollapse: "separate"
                        }}
                        aria-label="simple table"
                  >
                        <TableHead>
                              <TableRow>
                                    {headers.map((header, index) => (
                                          <TableCell
                                                key={header}
                                                classes={{
                                                      root: classes.root,
                                                }}
                                                className={classes.sticky}
                                          >
                                          <div className={classes.cellContainer}>
                                          <div>{header}</div>
                                          <div>
                                          <ColumnFilters
                                                fetchEffect={fetchEffect}
                                                columnName={header}
                                                columnIndex={index}
```

```
                                          onFilterSelect={onFilterSelect}
                                    />
                                    </div>
                                    </div>
                              </TableCell>))}
                        </TableRow>
                  </TableHead>
            <TableBody>
                  {rows.map((row, index) => (
                        <TableRow
                        key={index}
                        className={classes.tableRow}
                        >
                              {Object.keys(row).map(cell => (
                              <TableCell
                                    key={cell}
                                    component="th"
                                    scope="row"
                                    classes={{
                                          root: classes.root,
                                    }}
                              >
                                    <div>{row[cell]}</div>
                              </TableCell>
                  ))}
                        </TableRow>
                  ))}
            </TableBody>
      </Table>
</TableContainer>
);
}

CommonTable.propTypes = {
      fetchEffect: PropTypes.func,
      headers: PropTypes.arrayOf(PropTypes.string),
      rows: PropTypes.arrayOf(PropTypes.object),
      onFilterSelect: PropTypes.func
}
```

Лістінг error-message.component.js

```
import PropTypes from "prop-types";

import { useStyles } from "./error-message.styles";

export const ErrorMessage = ({ error }) => {
      const classes = useStyles();

      return (
            <div className={classes.errorMessage}>
            {error !== null && (
                  <div>
                        <h5>Error code: {error.errorCode}</h5>
                        <p>Error message: {error.errorMessage}</p>
                  </div>
            )}
      </div>
      );
}

ErrorMessage.propTypes = {
      error: PropTypes.exact({
      errorCode: PropTypes.number,
      errorMessage: PropTypes.string
      })
}
```

Лістінг nested-list.component.js

```
import React, { useState, useEffect } from "react";
import PropTypes from "prop-types";
import ListSubheader from "@material-ui/core/ListSubheader";
import List from "@material-ui/core/List";
import ListItem from "@material-ui/core/ListItem";
import ListItemIcon from "@material-ui/core/ListItemIcon";
import ListItemText from "@material-ui/core/ListItemText";
import Collapse from "@material-ui/core/Collapse";
import InboxIcon from "@material-ui/icons/MoveToInbox";
import ExpandLess from "@material-ui/icons/ExpandLess";
import ExpandMore from "@material-ui/icons/ExpandMore";
import StarBorder from "@material-ui/icons/StarBorder";
import {
      compose,
      curry,
      insert,
      remove,
      repeat,
      isEmpty
} from "ramda";
import classNames from "classnames";

import { ErrorMessage } from "@shared/error-message";
import { useStyles } from "./nested-list.styles";

export const NestedList = ({
      items,
      onItemClick,
      error
}) => {
      const classes = useStyles();

      const [open, setOpen] = useState(repeat(false, items.length));
      const [selected, setSelected] = useState({});

      useEffect(() => {
            if (!isEmpty(items)) {
                  setSelected({
                        database: items[0].title,
                        collection: items[0].children[0]
                  });
                  handleSectionClick(0);
            }
      }, [items]);
```

```
      const handleSectionClick = i => {
            setOpen(
                  compose(
                        curry(insert(i, !open[i])),
                        curry(remove(i, 1))
                  )(open)
            );
      };

      const handleItemClick = (title, child) => {
      const item = {
            database: title,
            collection: child
      };
      setSelected(item);
      onItemClick(item);
};

return (
      <List
      component="nav"
      subheader={
            <ListSubheader
                  component="div"
                  className={classes.nestedListSubheader}
            >
            Collections
            </ListSubheader>
      }
      className={classes.root}
      >
<div className={classes.error}>
      <ErrorMessage error={error} />
</div>
            {items.map((item, index) => (
                  <div
                        key={item.title}
                        className="nested-list-item"
                  >
                  <ListItem
                        button
                        onClick={() => handleSectionClick(index)}
                  >
                        <ListItemIcon className={classes.itemIcon}>
                              <InboxIcon />
                              </ListItemIcon>
                              <ListItemText primary={item.title} />
                              {open[index] ? <ExpandLess /> : <ExpandMore />}
                              </ListItem>
```

```
                              <Collapse
                                    in={open[index]}
                                    timeout="auto"
                                    unmountOnExit
                              >
                              <List
                                    component="div"
                                    disablePadding
                              >
                                    {item.children.map(child => (
                                    <ListItem
                                          key={child}
                                          button
                                          className={
                                                classNames(
                                                classes.nested,
                                          item.title === selected.database && child ===
                                    selected.collection ? classes.active : null
                                    )
                                    }
                                    onClick={() => handleItemClick(item.title, child)}
                              >
                        <ListItemIcon className={classes.itemIcon}>
                        <StarBorder />
                        </ListItemIcon>
                        <ListItemText primary={child} />
                              </ListItem>
                                    ))}
                              </List>
                              </Collapse>
                        </div>
                  ))}
            </List>
      );
}

NestedList.propTypes = {
      items: PropTypes.arrayOf(PropTypes.shape({
            title: PropTypes.string.isRequired,
            children: PropTypes.arrayOf(PropTypes.string)
      })).isRequired,
      onItemClick: PropTypes.func,
      error: PropTypes.shape({
      errorCode: PropTypes.number,
      errorMessage: PropTypes.string
      })
};
```

Лістінг core-table.component.js

```
import React, { useState, useEffect } from "react";
import PropTypes from "prop-types";
import Paper from "@material-ui/core/Paper";
import { omit, merge, curry } from "ramda";

import { CommonTable } from "@shared/common-table";
import { CommonPaginator } from "@shared/common-paginator";
import { ErrorMessage } from "@shared/error-message";
import { CommonSpinner } from "@shared/common-spinner";
import { useFetch } from "@core/hooks";
import { fetchTableData } from "./core-table.data";
import { useStyles } from "./core-table.style";

export const CoreTable = ({
      dataPath,
      pagesizeOptions,
      pagesize,
      page,
      handleChangePage,
      handleChangePagesize
}) => {
      const classes = useStyles();

      const [headers, setHeaders] = useState([]);
      const [rows, setRows] = useState([]);
      const [count, setCount] = useState(0);
      const [notFilter, setNotFilter] = useState({});
      const [loading, setLoading] = useState(true);
      const [error, setError] = useState(null);

      useEffect(() => {
            useFetch(
                  fetchTableData(
                  dataPath,
                  page + 1,
                  pagesize,
                  {},
                  { "$not": notFilter }
            ),
                  setLoading,
                  setError
            )
                  .then(res => {
                        setHeaders(Object.keys(res.elements[0] || []));
                        setRows(res.elements);
                        setCount(res.totalElements);
```

```
                })
            .catch(e => e);
    }, [dataPath, page, pagesize]);

    const onFilterSelect = (column, items) => {
        const filter = items.length === 0 ?
            omit([column], notFilter) :
            merge(notFilter, { [column]: items });
            setNotFilter(filter);
            useFetch(
                fetchTableData(
                    dataPath,
                    page + 1,
                    pagesize,
                    {},
                    { "$not": filter }
                ),
                setLoading,
                setError
            )
            .then(res => {
                setHeaders(Object.keys(res.elements[0] || []));
                setRows(res.elements);
                setCount(res.totalElements);
            })
            .catch(e => e);
    };

return (
    <Paper className={classes.root}>
        <ErrorMessage error={error}></ErrorMessage>
        <div className={classes.tableContainer}>
            <CommonSpinner
                loading={loading}
                size={16}
            >
        </CommonSpinner>
        <div className="tableWrapper">
            <CommonTable
                fetchEffect={curry(fetchTableData)(dataPath)}
                headers={headers}
                rows={rows}
                onFilterSelect={onFilterSelect}
            >
        </CommonTable>
        </div>
            <div className="paginatorWrapper">
                <CommonPaginator
                    pagesizeOptions={pagesizeOptions}
```

```
                              count={count}
                              pagesize={pagesize}
                              page={page}
                              handleChangePage={handleChangePage}
                              handleChangePagesize={handleChangePagesize}>
                        </CommonPaginator>
                  </div>
            </div>
      </Paper>
      );
};

CoreTable.propTypes = {
      dataPath: PropTypes.string.isRequired,
      pagesizeOptions: PropTypes.arrayOf(PropTypes.number),
      pagesize: PropTypes.number.isRequired,
      page: PropTypes.number.isRequired,
      handleChangePage: PropTypes.func,
      handleChangePagesize: PropTypes.func
}
```

Лістінг App.js

```
import React, { useState, useEffect } from "react";
import axios from "axios";

import { CoreTable } from "@core/components/core-table";
import { Header } from "@core/components/header";
import {
      UrlsConfig,
      getConfigUrls,
      createBackendAdapter
} from "@core/mock-backend";
import { NestedList } from "@shared/nested-list";
import { useStyles } from "./App.styles";

const App = () => {
const classes = useStyles();

      const adapter = createBackendAdapter();
      adapter()
            .initConfig(getConfigUrls(UrlsConfig))
            .initAdapter()
            .initGlobalMethods();
      const [dataPath, setDataPath] = useState("api/database1/collection1");
      const [page, setPage] = useState(0);
      const [pagesize, setPagesize] = useState(10);
      const [items, setItems] = useState([]);
      const [error, setError] = useState(null);
      useEffect(() => {
            axios.get("api/collections", {})
                  .then(response => {
                        return response.data.elements;
                  })
                  .then(response => {
                        setItems(response);
                        setError(null);
                  }).catch(err => {
                        setError({
                              errorCode: err.response.status,
                              errorMessage: err.message
                        });
                  });
            }, []);

            const pagesizeOptions = [5, 10, 15, 20];
            const handleChangePage = ($event, page) => {
```

```
            setPage(page);
      };
      const handleChangePagesize = $event => {
            setPagesize(parseInt($event.target.value, 10));
            setPage(0);
      }

      const onItemClick = item => {
            setDataPath(`api/${item.database}/${item.collection}`);
      };

      return (
            <div className={classes.app}>
                  <div className={classes.uiHeader}>
                        <Header items={items} />
                  </div>
                  <div className={classes.uiContainer}>
                  <NestedList
                        items={items}
                        onItemClick={onItemClick}
                        error={error}
                  />
                  <CoreTable
                        dataPath={dataPath}
                        pagesizeOptions={pagesizeOptions}
                        pagesize={pagesize}
                        page={page}
                        handleChangePage={handleChangePage}
                        handleChangePagesize={handleChangePagesize}>
                  </CoreTable>
            </div>
      </div>
      );
}

export default App;
```

ДОДАТОК Б

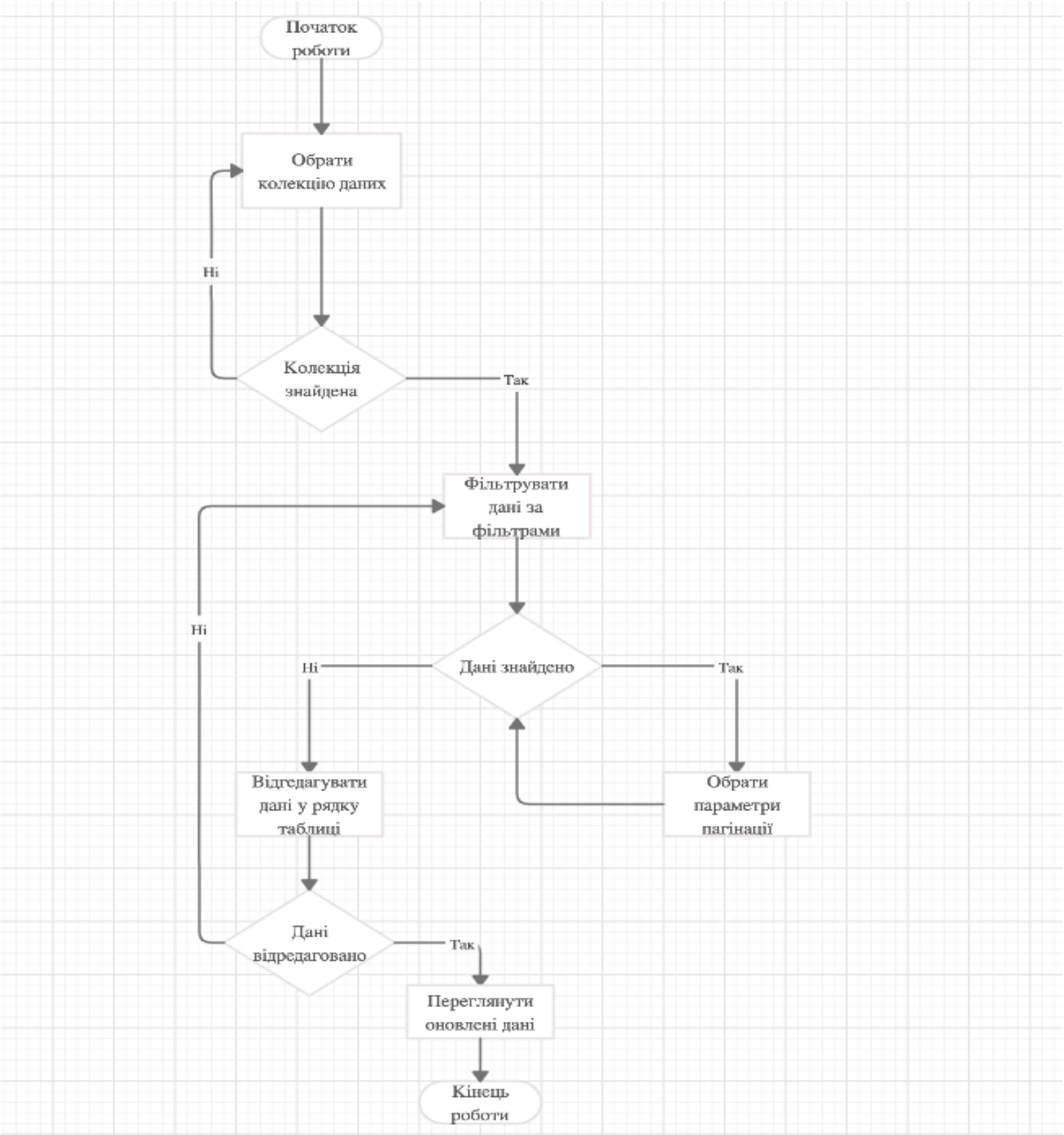
Система керування контентом

Блок-схема алгоритму

ІАЛЦ.466521.005 Э1

Аркушів 1

Київ — 2020 року



					ІАЛЦ.467800.006 Э1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Литвинчук Д.К.			Система керування контентом Схема функціональна Блок-схема алгоритму	Літ. / Аркуш / Аркушів
Перевірив		Корочкін О.В.				
						НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІП-62
Н. контр		Сімоненко В.П.				
Затверд.		Стіренко С. Г.				

ДОДАТОК В

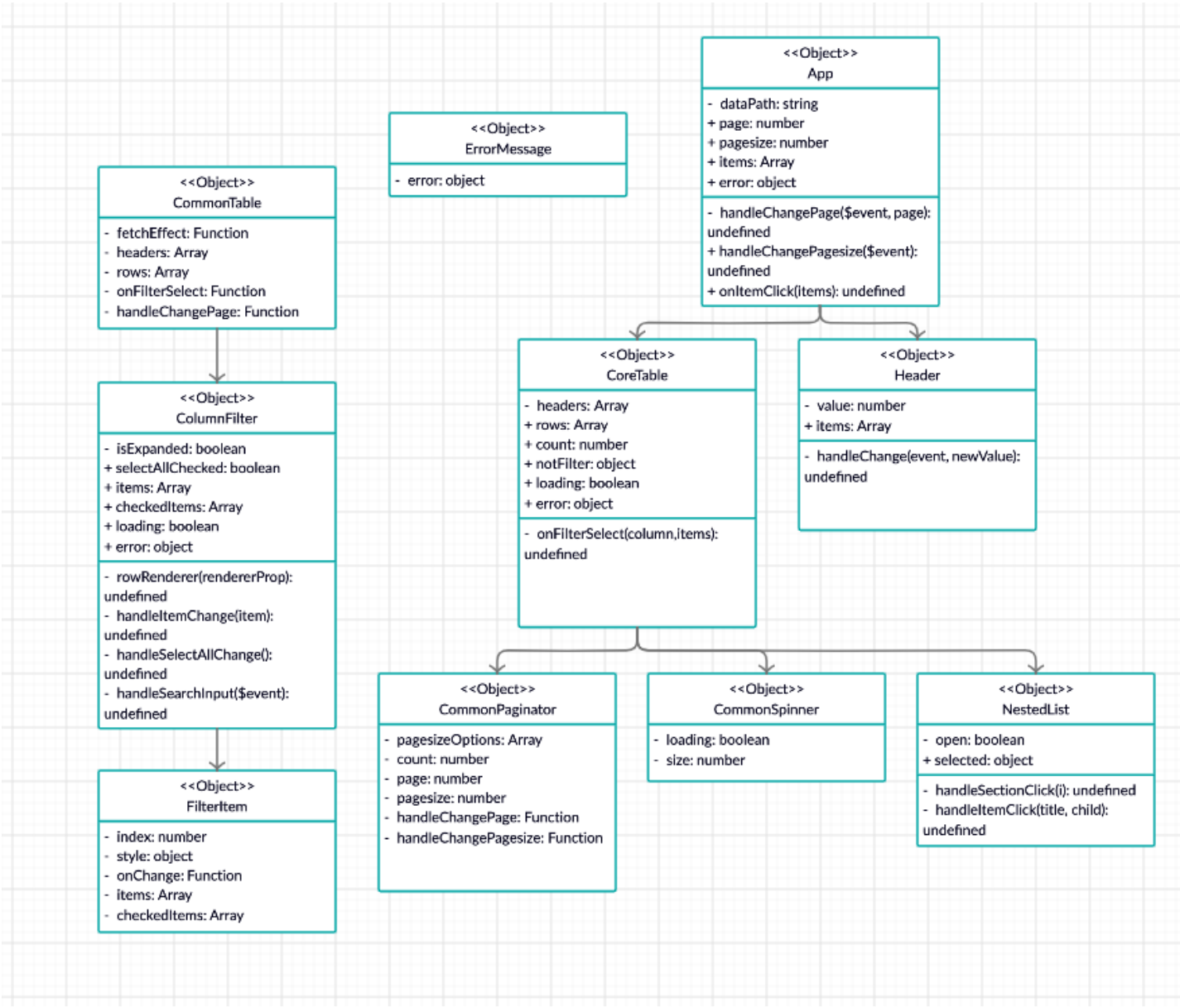
Система керування контентом

Діаграма класів компонентів

ІАЛЦ.466521.005 Д2

Аркушів 1

Київ — 2020 року



					ІАЛЦ.467800.007 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Литвинчук Д.К.			Система керування контентом Схема структурна Діаграма класів компонентів	Літ. / Аркуш / Аркушів
Перевірив		Корочкін О.В.				
						НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІП-62
Н. контр		Сімоненко В.П.				
Затверд.		Стіренко С. Г.				

ДОДАТОК Г

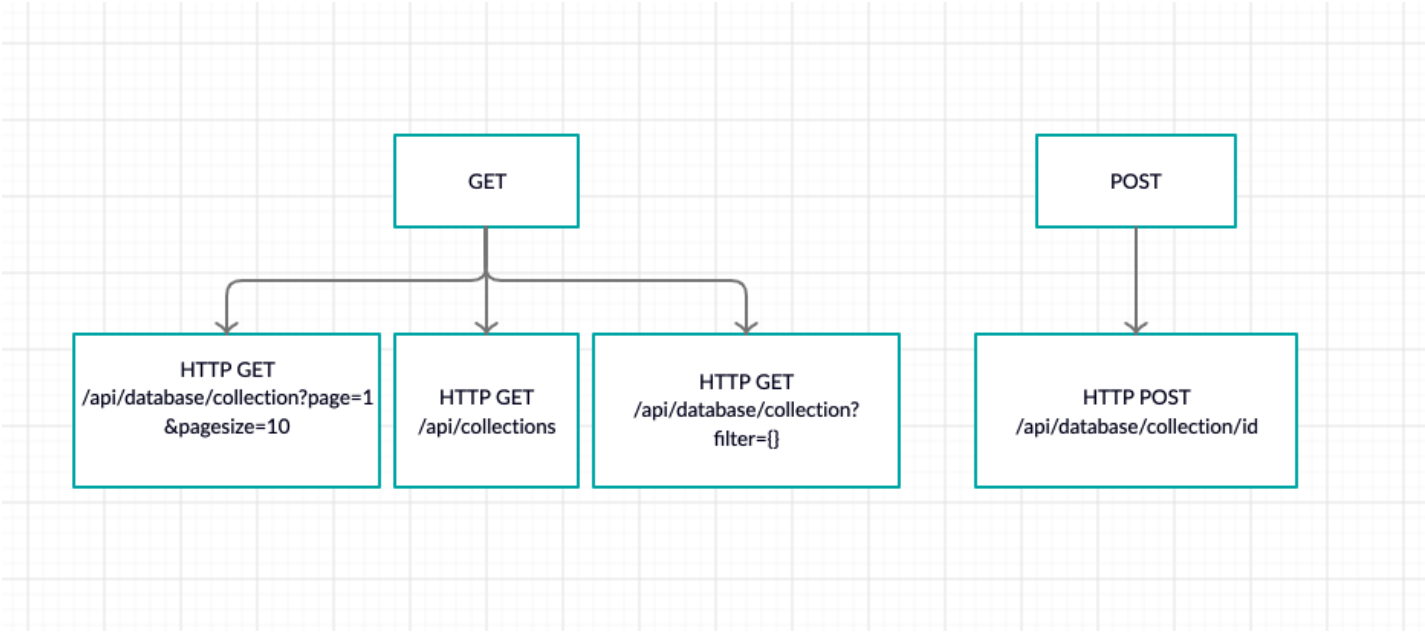
Система керування контентом

Діаграма класів запитів

ІАЛЦ.466521.005 Д3

Аркушів 1

Київ — 2020 року



					ІАЛЦ.467800.008 Д3		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Литвинчук Д.К.			Система керування контентом Схема структурна Діаграма классів запитів	Літ.	Аркуш / Аркушів
Перевірив		Корочкін О.В.					
						НТУУ «КПІ ім. Ігоря Сікорського» ФІОТ Група ІП-62	
Н. контр		Сімоненко В.П.					
Затверд.		Стіренко С. Г.					